Executive Summary

Wireless sensor networks (*WSNs*) are homogeneous application-centric networks composed of several resource-constrained devices and recently the integration with the traditional *IP* networks, using the *IPv6* protocol and *6LoWPAN,* have been demonstrated. Thus, *6LoWPAN* will be part of the evolution of the Internet. Therefore, the need for secure end-to-end communication between sensor nodes and Internet host is increasing, since sensing applications are connected to the Internet. Consequently, appropriate security techniques must be adopted in the context of *6LoWPAN*. This project introduces an in-depth evaluation of the performance of *IPsec* implementation in *6LoWPAN* networks of resource-constrained embedded devices and investigates if other techniques can be efficiently used, as well.

A detailed background research about the traditional *WSN*, *6LoWPAN, IPv6* and *IPsec* was performed in order to understand the state of the art technology in this area. Moreover, a review about the various methods that have been used by other researchers, providing end-to-end security on either the traditional *WSNs* or the *6LoWPANs* was a necessity, in order to examine if any other method, can be efficiently used in *6LoWPAN*.

An already existing *IPsec* implementation for *6LoWPAN* is extended and extensively evaluated, using *Cooja* simulator, which is distributed as part of the *Contiki OS*. Furthermore, *TLS/DTLS* were widely studied in order to be able to examine and optimize an already existing *DTLS* implementation in *6LoWPAN*.

As part of this project I completed the following tasks:

1. An extension of an already existing *IPsec* implementation in *6LoWPAN.*
2. An experimental evaluation of the extended *IPsec* implementation using *Cooja* simulator.
3. An investigation of the end-to-end delay and energy consumption overheads, packet loss and memory requirements under the extended *IPsec* implementation.
4. An examination and optimization of an already existing *DTLS* implementation in *6LoWPAN.*
5. An investigation of the optimized *DTLS* implementation in terms of memory footprints.
6. A comparison of *IPsec* and *DTLS.*

Table of Contents

1. Introduction and Scope

1.1 Introduction

Wireless sensor networks (*WSNs*) are homogeneous application-centric networks. They consist of several resource-constrained devices, in terms of network bandwidth, processing power and energy, and they are utilized to monitor and gather information about physical or environmental conditions, such as vibration, pressure, temperature or motion [1].

In the beginning, the research about *WSNs* suggested that *TCP/IP* protocols could not be used due to the aforementioned resource constrains. Therefore, the first *WSN* implementations were purpose design, such as the model presented in [2] by Estrin *et al.*

A number of recent research attempts, such as [3] and [4], have shown that *WSNs* can be used efficiently with layered, *IPv6*-based network architectures. That was the beginning for the growing of Internet standards which focus on the transmission and routing of *IPv6* datagrams over low-power and lossy *IEEE 802.15.4* radio links which is known as *6LoWPANs*. Sensor nodes in *6LoWPAN* can directly forward their collected data towards *IPv6* enabled hosts and for instance, data processing can be done into a powerful server [5].

One requirement of the real-world deployments of *WSNs,* is to establish an end-to-end secure data transmission. For example, an application for temperature or pressure measure would need a secure communication between the provider and the receiver. Provider and receiver would need to authenticate each other and data confidentiality should be provided using data encryption. Nevertheless, many research designs concentrate on hop-by-hop secure communications due to the fact that it is easier to be established, compare to end-to-end security. The reason for this is the two following intuitive approaches about how end-to-end security can be achieved in *WSNs* [6] :

- End-to-end can be accomplished through a secure tunnel between the sensor node and the sink. Hence, each sensor should share a unique pairwise key with the sink that will be used for data encryption and decryption (Further discussion in Section 3.1).
- End-to-end security can be established using hop-by-hop encryption and decryption between neighboring sensor nodes of the network (Further discussion in Section 3.2).

In a multi-hop *WSN*, the first approach leads to an important limitation. The ability of data aggregation, which helps in energy saving, is removed as for the intermediate nodes are not able to encrypt and decrypt the transferred data. In order to achieve the second approach, the links of the network must be high resilient, which is difficult in randomly deployed *WSNs*.

Even though the computational capacity of modern sensor hardware devices is limited, they are often equipped with an encryption co-processor, aiming to achieve secure transmission of network frames without consuming micro-controller (*MCU*) cycles. Thus, it would be beneficial for *6LoWPAN* to take advantage of this capability. Therefore, the end-to-end secure communication area for *6LoWPAN* is a new interesting and challenging research area that has to be explored.

1.2 Aims and Objectives

According to some research evidence Internet Protocol Security (*IPsec*) can be applied on *6LowPAN*, but the exact performance and the energy consumption implications are not known. Thus, this project has two aims. Firstly, it aims to conduct an in-depth evaluation of the performance of *IPsec* in *6LowPAN* networks of resource-constrained, embedded devices (8-bit or 16-bit MCU, 8-10 KB of RAM). Secondly, it aims to investigate if end-to-end security can be efficiently implemented using other appropriate technique. Hence, the two research questions which will guide the research are:

1. How *IPsec* performs in *6LowPAN* networks of resource-constrained, embedded devices (8-bit or 16-bit *MCU*, 8-10 KB of *RAM*)?

2. Is it possible to efficiently implement end-to-end security in *6LoWPAN* utilizing other appropriate methods except *IPsec*?

More specifically, the objectives are:

1. To extend an already existing *IPsec* implementation in *6LoWPAN*.

2. To investigate the end-to-end delay and energy consumption overheads under the extended *IPsec* implementation.

3. To investigate some other metrics such as memory requirements and packet loss.

4. To find another efficient method that could be applied on *6LoWPAN* providing end-to-end security.

5. To implement or extend this technique, using the Contiki Embedded Operating System [7].

6. To investigate the performance of the chosen method.

7. To compare the two methods.

8. To make experimental evaluation using simulators (*Cooja* simulator which is distributed as part of the *Contiki OS*).

2. Background

2.1 6LoWPAN vs Traditional WSN networking environment

2.1.1 Introduction

As presented in Figures 1 and 2, *6LoWPAN* utilizes the general protocol stack that is used in the general network (application layer, transport layer, network layer, *MAC* layer and physical layer) in contrast to the traditional *WSN,* that uses a protocol stack that consists of application, application-dependent network protocol and physical layer. We observe that *6LoWPAN* needs only the network protocol and lower layers, for data transmission compared to traditional *WSN* in which each node act as a router and all the layers are needed to relay data packets.

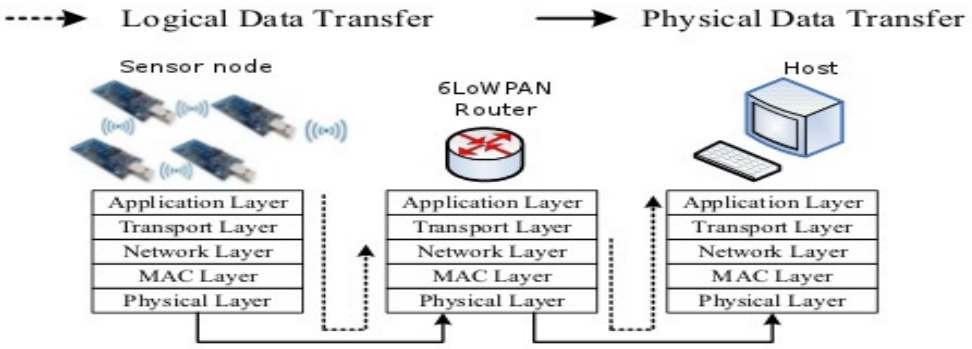


Figure 1: 6LoWPAN networking environment. It uses the general protocol stack with application layer, transport layer, network layer, MAC layer and physical layer. (From [8].)

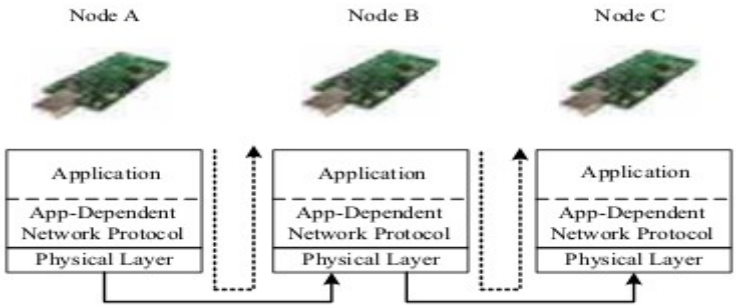


Figure 2: Traditional WSN networking environment. It uses a protocol stack consists of application, application-dependent network protocol and physical layer. (From [8].)

2.1.2 Traditional Wireless Sensor Networks

WSNs are application-centric networks that contain several resource-constrained sensor nodes and a sink. They are homogeneous devices and mostly identical from a hardware and software point of view. These nodes can sense and gather information about physical or environmental conditions, such as temperature, pressure, sound or humidity and send them to the sink. Moreover, *WSNs* utilize the *IEEE 802.15.4* standard which is defined by *IEEE* for low data rate wireless personal area networks. In accordance with Figure 2, different networking protocols should be able to be chosen in each layer without causing any problems to the other layers. This characteristic is important because different network protocols are utilized in various applications, since *WSNs* are application-centric networks. Furthermore, most of the applications use both multi-hop and hop-to-hop networking to transfer the data [8] [1]. An illustration of *WSN* is shown in Figure 3.

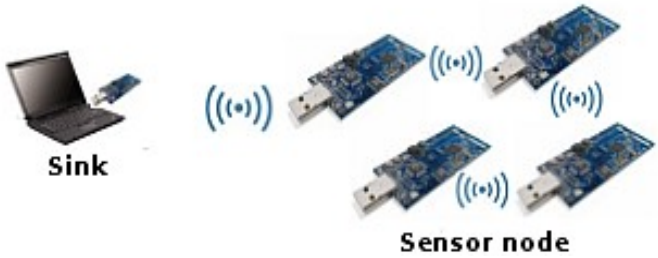


Figure 3 : Wireless Sensor Network with sensor nodes and a sink. (From [5].)

2.1.3 6LoWPAN

Internet Engineering Task Force (*IETF*), provided a set of standards that combine *IPv6* and Low Power Wireless Premise Area Networks, called *6LoWPAN,* over an *IEEE 802.15.4* network [9]. Shelby Z. and Bormann C. give a definition of *6LoWPAN* in [10]:

> "*6LoWPAN* standards enable the efficient use of *IPv6* over low-power, low-rate wireless networks on simple embedded devices through an adaptation layer and the optimization of related protocols."

This notion started from the idea that instead of trying to build a new protocol for small devices, Internet Protocol (*IP*) should be utilized . Using the *IP* in these applications and integrating them with the Internet of Things leads to many benefits [10]:

- Obviate the need for translation proxies or gateways when *IP*-based devices want to connect to other *IP* networks.
- Existing network infrastructure can be used in *IP* networks.
- There are many *IP*-based technologies which are well known and have been confirmed to work and scale.
- *IP* technology is better understood by a wider audience and encourages innovation, since the processes it uses are standards which are free and available to anyone.
- There are many tools for commissioning, managing and diagnosing *IP*-based networks.

Figure 4: 6LoWPAN Network. Sensor nodes are part of the existing IP based infrastructure and communicate with the IPv6 protocol (From [5].)

However, header compression mechanisms must be used in order to be able to use *IPv6* in *6LoWPAN*. This must be done, because of the fact that a maximum packet size in a physical layer of an *802.15.4* packet is *127* bytes and the maximum frame header size is *25* bytes. Therefore, an *IPv6* packet has to fit in *102* bytes. In addition, packet headers need the *48* bytes of the *102* remaining bytes [5] [10]. Context aware header compression mechanisms are defined in [11]. These are the *LOWPAN_IPHC* encoding format for *IPv6* header compression, which is illustrated in Figure 5, the *LOWPAN_NHC* for compressing the next header

and the *LOWPAN_NHC* Extension Header Encoding for *IP* extension headers which is illustrated in Figure 6. Some *IPv6* header fields are implicitly known to all nodes in the *6LoWPAN* network and therefore, they are not needed in *IP* packets. Thus, *LOWPAN_IPHC* removes them. Moreover, the length of the *LOWPAN_IPHC* is 2 byte, of which 13 bits are utilized for the compression. Any information from the uncompressed *IPv6* header fields follow the *LOWPAN_IPHC* encoding exactly as they would be in the normal *IPv6* header. In the best case, *IPv6* header can be compressed to 2 bytes with link-local communication and 7 bytes in a multi-hop scenario. To define if the next header, which follows the basic *IPv6* header is encoded or not, the *NH* field is used and if *NH* is equal to 1, the *LOWPAN_NHC* is utilized to compress it. The size of *LOWPAN_NHC* is defined by *6LoWPAN* to be multiple of octets and usually is 1 byte. From this byte the first variable length bit corresponds to a *LOWPAN_NHC ID* and the other bits are utilized to encode headers. Furthermore, *UDP* and *IP* Extension Header are specified by *6LoWPAN* as well. *LOWPAN_NHC EH* encoding contains a *NHC* octet, and 3 of them are utilized for the encoding of the *IPv6* Extension Header *ID* (*EID*).

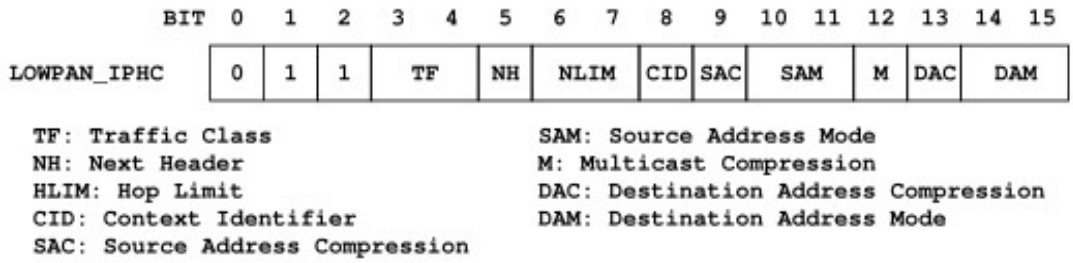


Figure 5: The LOWPAN_IPHC Header (From [5].)

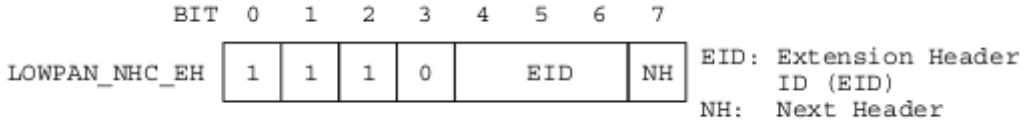


Figure 6: The LOWPAN_NHC_EH encoding for IPv6 Extension Header (From [5].)

2.2 IPv6

2.2.1 Introduction

Internet Protocol version 4 (*IPv4*) [12] has limited address space, since it handles only *32* bit addresses and in the near future it will probably runs out of the available space, if we consider all the new mobile devices that will be connected to the Internet [13]. Moreover, *IPv4* has lack of security and therefore a new *IP* version that would reduce or remove these limitations was needed. Internet Protocol version 6 (*IPv6*) [14] provides new features that deal with the above problems, such as the increase of the address size from *32* bits to *128* bits. In addition *IPv6* has a simplified header format, improved support for extensions and options, flow labeling capability, and authentication and privacy capabilities. *IPv6* header, that is shown in Figure 7, contains only the fields with the functionalities that are required for all the packets. If more functionalities are needed, then some extension headers are implemented and are placed between the *IPv6* header and the upper layer header in the packet. As a result, the processing cost of packet handling is reduced and the bandwidth cost of the *IPv6* header is limited. An *IPv6* packet may contain zero or several extension headers and each one can be identified by the Next Header field of the preceding header [9] [13].

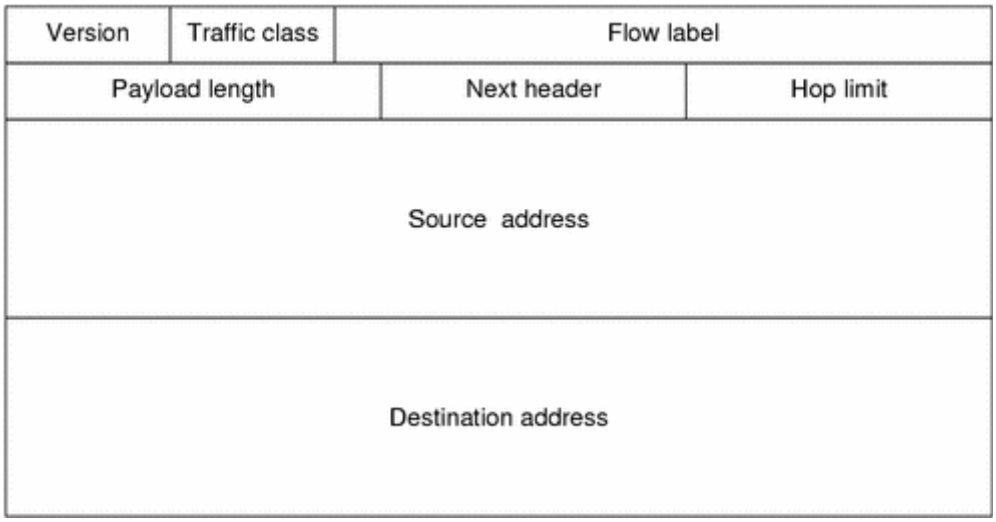


Figure 7: IPv6 Header Format[1]

The recommendation order of extension headers in a packet is the following [14]:

- IPv6 header
- Hop-by-Hop Options header
- Destination Options header (for options that will be used by the first destination that appears in the *IPv6* Destination Address field and other destinations listed in the Routing header)
- Routing header
- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header (for options that will be used only by the final destination of the packet)
- Upper-layer header

2.2.2 IPsec

2.2.2.1 Introduction

Security (authentication and privacy) in *IPv6* is provided by the Internet Protocol Security (*IPsec*) framework [15] which provides network-level security. *IPsec* is an end-to-end security scheme which utilizes a set of protocols in order to secure *IP* communications. These are the Authentication Header (*AH*) [16] and the Encapsulating Security Payload (*ESP*) [17] security protocols, the key exchange mechanisms, the algorithms for authentication and encryption and the security associations (*SA*) [15]. In addition, *IPsec* has two basic modes of use, a transport mode and a tunnel mode. In the transport mode, *IPsec* processing is performed at endpoints of secure channel and *IP* header and payload are directly secured. In the tunnel mode, the original *IP* datagram is encapsulated within a new *IP* header, at which security functions are applied and the encrypted datagram is not visible to intermediate routers. *IPsec* processing is performed at security gateways on behalf of endpoint hosts.

2.2.2.2 Authentication Header

Authentication Header is shown in Figure 8 and provides data origin authentication and connectionless integrity for *IP* datagrams. Data origin authentication means that if a computer or another device connected

1 From <http://docs.oracle.com/cd/E19683-01/817-0573/chapter1-fig-8/index.html>.

to the Internet receives an *IP* datagram, it can be assured that this packet came from the source address written on the *IP* header. Connectionless integrity means that if a computer or another device connected to the Internet receives an *IP* datagram, it can be assured that the contents of this packet have not been modified during the transmission from source to destination. Furthermore, using *AH*, protection against replays attacks is provided [13]. Data authentication is produced by a keyed Message Authentication Code (*MAC*) that is applied to the *IP* header, *AH* header and *IP* payload. The hash-based message authentication code algorithm *AES-XCBC-MAC-96* must be supported from all the hosts in order to be able to calculate authentication data with size equal to 12 bytes [16].

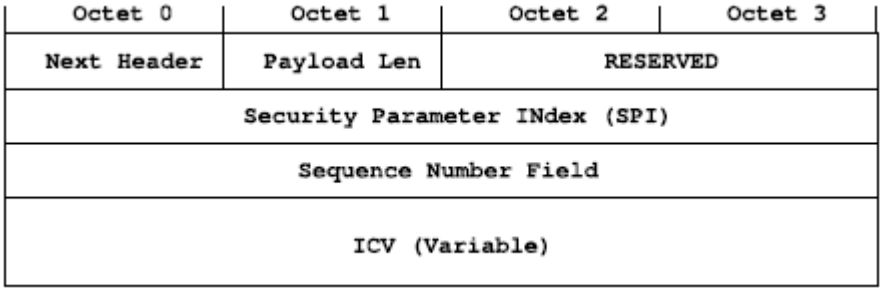


Figure 8: IPsec Authentication Header. (From [5].)

2.2.2.3 Encapsulating Security Payload

Encapsulating Security Payload is shown in Figure 9 and is utilized for confidentiality, data origin authentication and connectionless integrity. It can also provide (limited) traffic flow confidentiality and an anti-replay service. *ESP* encrypts the payload of an *IP* packet but the authentication algorithm is not applied to the *IP* header and therefore *IP* header is not protected. When the *ESP* is applied, the *ESP IP* extension header in which the encrypted payload exists, follows the *IP* header. Furthermore, *ESP* contains a Security Parameter Index (*SPI*) that identifies which algorithms and keys are to be used for *IPsec* processing, a sequence number used for the prevention of replay attacks, the encrypted payloads, optional authentication data and a reference to the next header. Some block ciphers require padding, thus *ESP* contains the required padding as well. Moreover, *ESP* uses symmetric encryption and *MACs* based on secret keys shared between endpoints. Encryption is applied to Payload Data, Padding, Padding Length and Next Header, and authentication if selected, is applied to all the header fields of *ESP* [17].

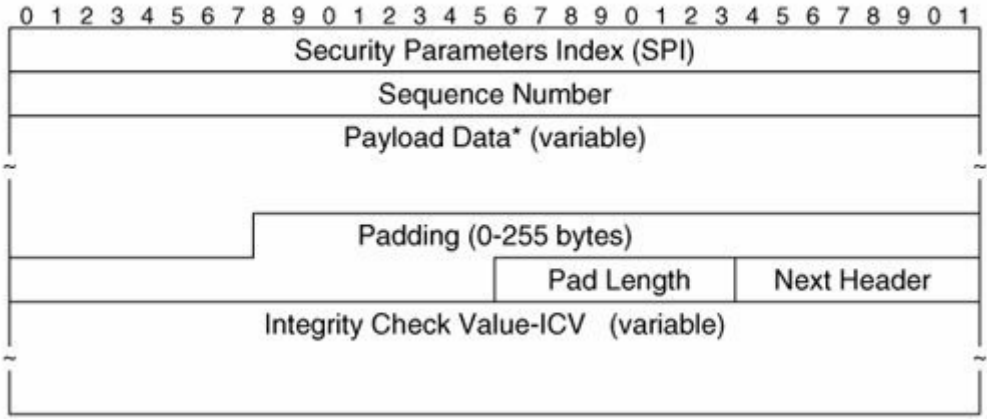


Figure 9: IPsec Encapsulation Security Protocol (From [17].)

2.2.2.4 Security Associations

Security association is a crucial concept in *IPsec* since it defines what type of security services a specific *IP* flow should have and usually contains the authentication and encryption algorithms to be used and the key needed for authentication or encryption. *SAs* are established in *IPsec*, using both pre-shared key and Internet Key Exchange (*IKE*) protocol. Therefore any *IPv6* node on the Internet supports pre-shared key. In contrast, *IKE* uses asymmetric cryptography which is heavy weight for small sensor networks [15].

2.3 Contiki Operating System

Contiki Operating System (*Contiki OS)* [7] is produced by the Swedish Institute of computer science and it is an open source operating system for the Internet of Things (*IoT*). It is lightweight and highly portable, and its memory usage is about *40 KB* of *ROM* and *2 KB* of *RAM.* That is why it is used for highly memory efficient embedded devices. The first version was released in 2004 and the latest version (version 2.6) was released in July 2012. It is implemented in *C* language and it is currently utilized on a set of low-power wireless hardware platforms such as *MSP430*, and *AVR*.

The biggest benefit of *Contiki* is that applications and services can be dynamically loaded and unloaded. This reinforces the resource usage and makes the kernel event driven.

2.4 Cooja

Cooja [18] [7] is the *Contiki* network simulator and is very useful for testing software, written for *Contiki*, before testing it in a real sensor network hardware. It is very hard to develop and debug software for large wireless networks. *Cooja* eliminates this difficulty by allowing developers to see their applications run in large-scale networks as well as in extreme detail on fully emulated hardware devices in its simulation environment.

2.5 Tmote Sky

Tmore Sky platform [19], illustrated in Figure 10, is an extremely low power wireless sensor board which is compatible with the *IEEE 802.15.4* standard. It is powered by *AA* batteries and also a *USB* port, that can be connected to a computer and derive power from it. It also consists of an *MSP430 F1611* microcontroller composed of *10 KB RAM* and *48 KB ROM* and it has the capability of fast wake up from sleep which takes less than 6μs. Furthermore, *Tmote Sky* consists of an integrated antenna with *50m* range indoors and *125m* outdoors, *1 MB* external serial flash memory, three integrated sensors (Temperature, Humidity and Light sensors) and a *Chipcon CC2420* radio chip that produces reliable wireless communication. In addition, it provides hardware link-layer encryption and authentication.

Figure 10: Tmote sky (From [19].)

3. Related Work

3.1 End-to-end security communication without data aggregation

End-to-end security is provided through a secure tunnel between two end points. Hence, the data are encrypted in the source point and the decryption can only be done by the destination point. End points in a *6LoWPAN* network are the sensor nodes and any other client of the Internet, e.g a server or a host. Therefore in a multi-hop network, such as the *6LoWPAN* the intermediate nodes between the source and the destination are not able to encrypt or decrypt the transferred data. As a result, data aggregation can not be achieved in networks with this security approach.

3.3.1 6LoWPAN with compressed IPsec

3.3.1.1 6LoWPAN IPsec with LOWPAN_NHC AH and ESP Encoding

Raza *et al.* in [5] proposed a compressed design, implementation and evaluation of *6LoWPAN* using *IPsec*. This implementation supports both *IPsec's AH*, that provide data integrity and authentication, and *ESP*, that provide confidentiality, integrity and authentication. Additionally, they show that the overheads of their implementation can be compared to overheads of the *802.15.4* link-layer security by evaluating the performance of the implementation in terms of code size, communication performance and packet overheads. Therefore, they show that *IPsec* is viable with *6LoWPAN* and provides true end-to-end security.

As mentioned above, *6LoWPAN* uses header compression mechanisms in order to utilize *IPv6* headers. Hence, compression mechanisms must also be used for the *IPsec*. The authors proposed extension header encodings for *AH* and *ESP* and they used the *2* free slots of the already defined *LOWPAN_NHC_EH* encoding in order to encode them. In addition, to determine that the next header (*AH* or *ESP*) is also encoded using *NHC*, they set the last bit of the *IPv6* extension header encoding to 1.

For the *AH* they define the *LOWPAN_NHC_AH* encoding that is shown in Figure 11 and each header field is described below:

- The first *4* bits are set to *1101* and represent the *NHC ID*.
- PL: Is used to define if the payload length is carried in-line after the *NHC_AH* header or not. If *PL* is equal to *0* then the payload length is omitted and if is equal to *1* is carried.
- SPI: Defines if the default *SPI* for the sensor network is utilized or not. If *SPI* is 0 then the default *SPI* is used and the *SPI* field is omitted, otherwise (if it is *1*) the *SPI* is carried in-line. They set the default *SPI* value to *1* which means that every node of the network has its own preferred *SA*.
- SN: Defines if all *32* bits of sequence number is carried in-line or only *16* bits of them. If it is equal to *0* then the left most *16* bits are assumed to be *0* and if it is equal to *1* all *32* bits are used.
- NH: Defines if the next header field in AH will be used or not. If *NH* is equal to *0* the next header field will be used and it is carried in-line, in order to determine the next header. Otherwise, (*NH* is equal to *1*) it will be skipped.

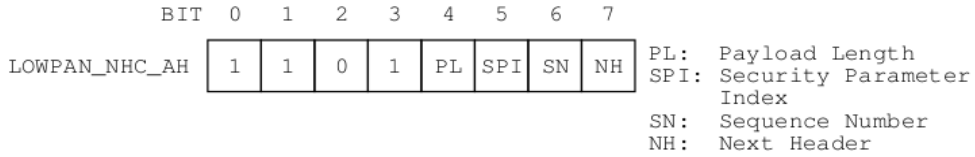


Figure 11: LOWPAN_NHC_AH: NHC encoding for IPv6 Authentication Header (From [5].)

A standard *AH* that uses the mandatory *HMAC-SHA1-96* needs a minimum of *24* bytes and after an optimal compression the header size is reduced to *16* bytes.

For the *ESP* header they define the *LOWPAN_NHC_ESP* encoding that is illustrated in Figure 12 and has the same fields as the *AH*. The two differences between *AH* and *ESP* fields is that in *ESP* the first *4* bits are set to *1110* compare to *AH* which are set to *1101* and that the default value of *SPI* is set to *0* in contrast to *AH* in which it is set to *1*. *ESP* without authentication, *AES-CBC* and perfect block alignment has *18* bytes overhead which is reduced to *12* bytes after an optimal compression, and the overhead of an *ESP* with authentication (*HMAC-SHA1-96*) is *30* bytes which can be reduced to *24* bytes after an optimal compression.

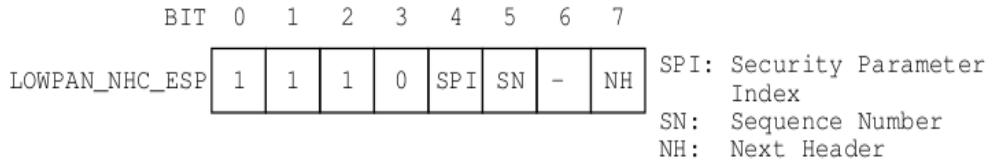


Figure 12: LOWPAN_NHC_ESP: NHC encoding for Encapsulating Security Payload header (From: [20].)

Figures 13 and 14 illustrate compressed *IPv6/UDP* packet using *AH* and *ESP* with *HMAC-SHA1-96* respectively.

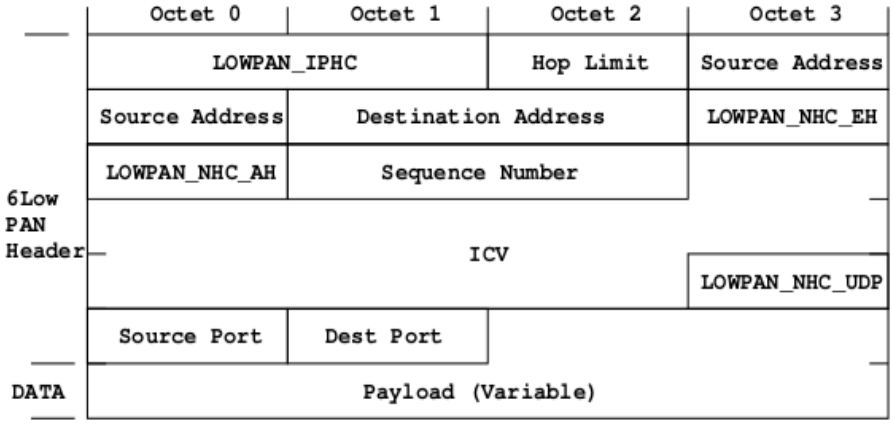


Figure 13: Compressed *IPv6/UDP* packet using *AH* with *HMAC-SHA1-96* (From [5].)

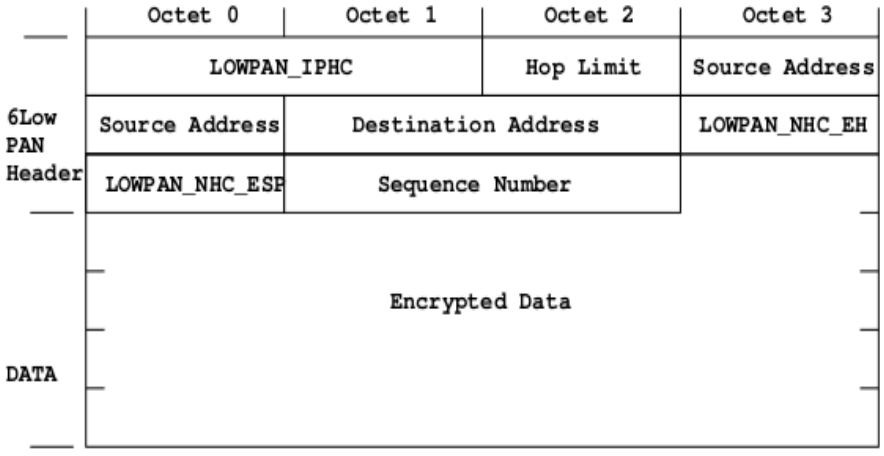


Figure 14: Compressed *IPv6/UDP* packet using *ESP* with *HMAC-SHA1-96* (From [20].)

In order to test their implementation, they used the *Contiki OS* [7] with pre-shared keys. They estimate the impact of *IPsec* in terms of memory footprint, packet size, energy consumption and performance under different configurations.

For memory footprint, they measured the *ROM* and *RAM* footprint of their implementation. and they found that the *ROM* footprint overhead ranges from *3.8 KB* to *9 KB* and the *RAM* footprint overhead ranges from *0.3 KB* to *1.1 KB*. Therefore, we can see that both *IPsec AH* and *ESP* can be used in resource-constrained devices.

Table 1 makes a comparison of the packet overhead when uncompressed *IPsec*, compressed *IPsec* and *802.15.4* link-layer security is used. As we can see, the overhead when using compressed *IPsec* is not that larger from the overhead when using *802.15.4* link-layer security. Furthermore, for the transmission of large *IP* datagrams, link-layer fragmentation is needed. Thus, when using link-layer security the overhead will exist in every fragment, in contrast to *IPsec* where the *IPsec* header exists only once for all the fragments. As a result, when fragmentation is used, *IPsec* has lower header overhead than *802.15.4* link-layer security.

Service	Uncompressed IPsec		Compressed IPsec		802.15.4	
	Mode	Bytes	Mode	Bytes	Mode	Bytes
AH Authentication	HMAC-SHA1-96	24	HMAC-SHA1-96	16	AES-CBC-MAC-96	12
ESP Encryption	AES-CBC	18	AES-CBC	12	AES-CTR	5
ESP Encryption and Authentication	AES-CBC and HMAC-SHA1-96	30	AES-CBC and HMAC-SHA1-96	24	AES-CCM-128	21

Table 1: Comparison of packet overhead with uncompressed IPsec, compressed IPsec and 802.15.4 link-layer security. Packet sizes are similar in compressed IPsec and in 802.15.4 security. (From [5].)

The researchers, compared the efficiency of different cryptographic algorithms and modes that their implementation supports. The results showed that *AES-CBC* and *AES-XCBC-MAC-96* are the most efficient in terms of energy consumption and processing time.

Moreover, they measured the energy overhead by calculating the total number of *CPU* ticks from the time the first fragment of the message is gathered at link-layer decryption until the end of the last packet's encryption. Due to the fact that they utilized both authentication and encryption for *ESP* , *AH* consumes less energy than *ESP*. They also noticed that the difference of the energy consumption when using *IPsec* and without *IPsec* is negligible, compared to the consumption of typical radio chips.

In addition, the authors estimated the response time for different data sizes when *IPsec* is used and when is not used. The response time is the time needed to send a message from an *IP* connected machine to a sensor node and take a response. *ESP* is faster than *AH* when the data sizes are small due to the fact that it does not process the *40* bytes *IP* header, like *AH*. In contrast, when the data sizes are large, *AH* is faster than *ESP*, because *AH* only ensures authentication compared to *ESP*, which authenticates, encrypts and decrypts the messages. Furthermore, the overhead of either *AH* or *ESP* is independent of the number of hops. They also observed that *IPsec's* efficiency can be enhanced with the use of cryptographic functions produced by sensor node hardware.

Even though a combination of *AH* and *ESP* can be used, this will not work on *6LoWPAN* as for, in this way the header overhead will be very large. Instead of this, it is better to use *ESP* with authentication options but a new problem arises from this option. When *ESP* is used, upper layer headers such as *UDP* can not be compressed because the encrypted *UDP* header can not be accessed and expanded from a *6LoWPAN* gateway between the *IP* and the sensor network. Therefore, a new *ESP* algorithm must be specified to perform this compression.

3.3.1.2 6LoWPAN IPsec with AH-HC and ESP-HC

Granjal *et al.* in [21] also developed an *IPsec* implementation for *6LoWPAN* that provides end-to-end security. They provided an *AH* and an *ESP* header named *AH HC* and *ESP HC* respectively, with similar functionalities as [5]. The authors, evaluated their implementation by measuring the available network payload with the *6LoWPAN* security headers, the processing time and the energy required for data encryption from real sensor nodes on specific *WSN* applications and the impact of security on data transmission rates.

In order to estimate payload space requirements, they considered *3* different compression scenarios. These scenarios are described below in an order from the scenario with the best usage, to the scenario with the worst usage, in terms of compression efficiency :

a) Compression with unicast link-local communications.
b) Compression with communications outside of link-local scope.
c) Compression with communications outside the *LoWPAN*.

In their experiments for evaluation of the processing time and energy requirements for data encryption, they used *AES*, *3DES*, *SHA1* and *SHA2*. They observed that *AES* needs less time than *3DES* for the *ESP* compression and the difference in energy usage between those two algorithms is negligible (*AES* needs *0.003512 mV/h* and *3DES* needs *0.0031 mV/h*). Moreover, *SHA1* is by far better than *SHA2* to provide authentication and integrity in respect to *AH*. They also proposed the *WSN* application security profiles, which are described in Table 2, that help in analysing the impact of security on sensor node lifetime. The lowest impact on sensor node lifetime for authentication with compressed *AH* is observed using *SHA2*. For compressed *ESP* they noticed that *3DES-SHA1* has better performance than *AES-CCM* and *3DES* with *SHA2*, but they considered that the best suite is the *AES/CCM*, since it is already available on sensor node hardware.

Furthermore, the authors using their previous results of the processing time and the energy required for data encryption, stated that applications which do not need confidentiality have higher transmission rates if they use *AH*. On the other hand, *ESP* is better to be used with applications with high security requirements or low data rates.

Application security profile	Integrity, authentication, non-repudiation	Confidentiality	6lowpan headers and algorithms
Intra-WSN monitoring	High	Not required	AH SHA2
Internet-WSN monitoring	Medium	Medium	ESP 3DES-SHA1
Inter-WSN control	High	High	ESP AES-CCM-128
Internet-WSN control	High	Medium	ESP 3DES-SHA2

Table 2 : WSN Application Security Profiles. (From [21].)

Conclusion and critical evaluation

As it can be seen from their results, authors in both papers managed to develop a viable *IPsec* implementation for *6LoWPAN* which can produce end-to-end secure communications. However, authors in

the first paper [5] mentioned that *ESP* with authentication options is better than *AH* and consequently, a new *ESP* algorithm has to be developed in order to do the compression that cannot be done if *ESP* is used. This leads to the problem that the new *ESP* algorithm should be included in all the *IPsec* hosts of the *6LoWPAN*. This constitutes a problem because *6LoWPAN* does not contain only sensor nodes and a sink like the traditional *WSNs*. Sensor nodes in *6LoWPAN* are part of the Internet and thus, this compression algorithm must be implemented in all the *IPsec* hosts of the Internet. Moreover, using *IPsec* in *6LoWPAN*, the ability for data aggregation, that helps in energy saving, is lost, because only the source and the destination node/host is able to encrypt and decrypt the transferred data.

3.1.2 TLS and DTLS for Contiki OS

Parelman V. and Ersue M. in [22] [23] [24] developed a *TLS* and a *DTLS* implementation for the *Contiki OS* that follows a pre-shared key approach and supports the *TLS_PSK_WITH_AES_128_CCM_8* and *DTLS_PSK_WITH_AES_128_CCM_8* [25] cipher suite accordingly. Those implementations do not support any of the optional messages (e.g. certificate-related messages) in order to be as small as possible. Figure 15 described the handshake protocol for those implementation.

According to the authors, both implementations are available as a *Contiki* internal library and includes functions for both, *TLS/DTLS* client and *TLS/DTLS* server, respectively. However, the application which includes *TLS/DTLS* can only act either as *TLS/DTLS* client or *TLS/DTLS* server, but not both simultaneously. *SHA256* and *HMAC_SHA256* are computed using an already existing external library [26] and for the *AES* operations they used libraries of the *OpenSSL* project with a little modification. Moreover, all the static variables used by the *AES* operations are stored in *Flash* memory, in order to reduce the amount of data that will be stored on *RAM* memory.

Furthermore, they stated that they tested their implementations on Raven boards [27] and they are working on *Contiki OS.*

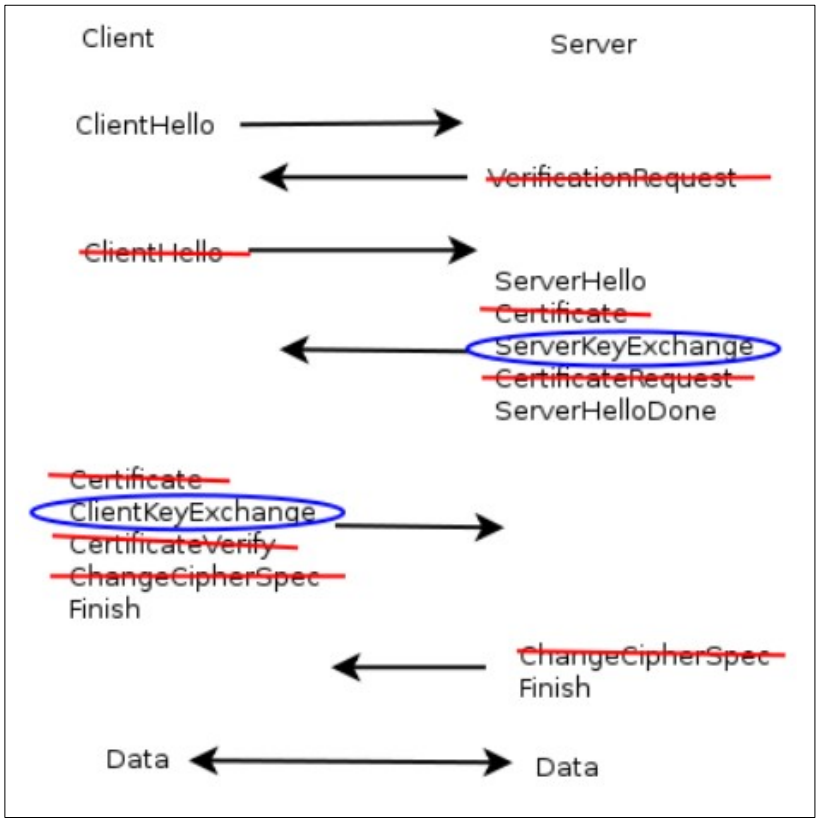


Figure 15: TLS and DTLS Handshake protocol in Contiki OS[2]

2 From <Korte K. D. , "DTLS for Contiki," Jacobs University Bremen , 2010>

Conclusion and critical evaluation

Authors proposed a lightweight implementation of *TLS* and *DTLS* protocols and stated that they have been tested on Raven board and it works. Nevertheless, they did not provide any of their results and thus, the performance of those implementations is not known. *TLS* and *DTLS* are widely spread into the Internet and thus it is crucial for *WSNs* to be able to interface with these protocols.

3.1.3 Security challenges in the IP-based Internet of Things

Heer *et al.* in [28] argued about the suitability and the constraints of the existing security algorithms and Internet protocols in the area of the Internet of Things (*IoT*). There are many security protocols for the Internet domain and authors chose to discuss about Internet Key Exchange (*IKEv2*)/*IPsec* [29], Transport Layer Security/Secure Sockets Layer (*TLS/SSL*) [30] , Datagram Transport Layer Security (*DTLS*) [31], Host Identity Protocol (*HIP*) [32] [33], Protocol for Carrying Authentication for Network Access (*PANA*) [34] and Extensive Authentication Protocol (*EAP*) [35] protocols. They also mentioned the Secure Shell (*SSH*) [36] protocol, but they did not analyse it further.

IKEv2/IPsec and HIP

They settle above or at the network layer and can secure the payload delivery by setting up the *IPsec* transforms and by achieving an authenticated key exchange. There are continuing endeavors nowadays to construct a variation of the *HIP*, namely *Diet HIP* [37], for the authentication and key exchange level of lossy low-power networks.

TLS and DTLS

They are used to provide security between applications, since they operate between transport-layer and application-layer. They have many similarities and use the same cipher suites. The difference between the two protocols is that *TLS* can only be used over *TCP*, compared to *DTLS* which can be used over *UDP*.

EAP

EAP resides at the data link-layer and supports various authentication methods. Moreover, *EAP* sustains retransmission and duplicate detection but it does not support packet fragmentation.

PANA

PANA behaves as a network-layer transport for *EAP* and is utilized to enabling network access authentication between the network infrastructure and clients.

Furthermore, authors advocated that even though the gap between Internet protocols and *IoT* is reduced, some differences still remain, which can be bridged using protocol translators at gateways. However, the described protocols produce *end-to-end* security that do not support translation at gateways.

Conclusion and critical evaluation

The authors elaborated on a number of existing security algorithms and Internet protocols and they supported that they can be suitable with *IoT* if protocol translators are used at gateways, something currently not supported. Therefore, in order to enable *end-to-end* security to *WSNs,* the protocols should be altered to support translation at gateways. In addition *TLS* and *DTLS* are rarely used in *WSNs* because they are considered “heavy weight”, thus it is better not to be used.

3.1.4 WirelessHART

Raza *et al.* in [38] presented an overview of *WirelessHART [39]* security. *WirelessHART* is a standard for *WSNs* developed for control systems and industrial process automation. It provides end-to-end security, per-hop security and peer-to-peer security but we are only interested in end-to-end security. Network layer provides end-to-end security and encrypts all the data that are passed to the link layer. These data can be decrypted only by the destination device.

The Network Protocol Data Unit (*NPDU*) of the *WirelessHART* has the following three fields:

- NPDU Header: It is utilized for data routing.
- Security Sub-layer: Consists of:
    - Message Integrity Code (*MIC*): Is used for data and source integrity between source and destination.
    - Counter: It is a four byte counter and used to create the nonce.
    - Security Control Byte: Defines the type of the security that is utilized.
- NPDU Payload: It is a *Transport Layer PDU* (*TPDU*) and is always encrypted using *AES* with *128* bit key.

Network Layer uses the *AES* in *Counter* with *CBC-MAC* (*CCM*) mode to calculate the *MIC* in order to provide data authentication and integrity, and to encrypt the *NPDU* payload in order to provide confidentiality. Moreover, network layer uses the same key to encrypt and calculate the *MIC*.

Conclusion and critical evaluation

The authors described the *WirelessHART* security that provides end-to-end, per-hop and peer-to-peer security. These security features can not be used in *6LoWPAN* because they can only be used with the *WirelessHART* network protocol, which is different from the Internet protocol that is used in *6LoWPAN*.

3.1.5 LEDS

Ren *et al.* in [40] provided a Location-aware End-to-End Data Security mechanism (*LEDS*) by integrating a location-aware key management framework and an end-to-end data security mechanism. They assumed a large scale uniformly distributed *WSN* that controls a large domain. The domain is virtually split into multiple cells and contains many static sensor nodes. *WSN* is a well connected network and is robust against node loss and failure. Moreover, the geographic location of each node can be obtained via a localization scheme and every event can be observed by various sensor nodes. *LEDS*, follows a one-to-many data forwarding approach exploiting the broadcast nature of the wireless links. When an event occurs, sensing nodes prepare a report and then transfer it, along a specific route to the sink, usually with multiple hops. Each event report is encrypted using the cell key of the analogous event cell and therefore data confidentiality is established. Moreover, to defend against the report disruption attack a pre-defined Linear Secret Sharing Scheme (*LSSS*) is used and splits the encrypted report into a number of unique shares. Furthermore, cell-to-cell authentication keys are utilized in order to deal with the selective forwarding attack. Finally, data authenticity is accomplished using both end-to-end verification at the sink and en-route filtering at the intermediate sensor nodes.

In addition, regarding to their performance analysis, sensor nodes store a small number of keys and this number is independent of the network size. This means that *LEDS* is feasible in large scale *WSNs*. Also, sink stores a small number of keys as well.

Conclusion and critical evaluation

Authors produced a mechanism which provides location aware end-to-end secure communications. This scheme is weak to fault-tolerance, since the sensing report transmission is achieved toward a specific route.

Hence, if a sensor node of this route is offline the data will be lost. Moreover, it cannot be used in the *6LoWPAN*, due to the fact that *6LoWPAN* uses the existing routing protocol of the *IP-based* infrastructure, since sensor nodes and Internet are combined in the same network. Another problem of this implementation is that the message event transmission is accomplished using broadcasting. This is dangerous to be implemented in *6LoWPAN*, because it is not safe to broadcast the data to all the hosts of the Internet that the source is connected with.

3.1.6 MASA

Alzaid H. and Alfaraj M. in [41] provided an end-to-end data security scheme in *WSNs* using Mixture of Asymmetric and Symmetric Approaches (*MASA*). They considered a large terrain, consisting of many low power sensor nodes, which is broken down into smaller cells. Each node can determine its geographic location through one of the various existing localization schemes, discover its neighboring nodes and compute its cell key. Moreover, each node has its own private key and only the sink has the corresponding public key. They also make the assumption that an event can only be sensed in one cell, and all the events are forwarded to the sink which is able to make powerful computations and has the appropriate memory to store all the public keys.

MASA is divided into 4 phases:

- Bootstrap Phase
- Generation Phase
- Forwarding Phase
- Verification Phase

Bootstrap Phase

Each sensor node calculates its cell key that will be used for authentication of the communication into the cell, and then its being recognised from its neighboring nodes, by broadcasting its existence to them. Each node has a list of trusted neighbors which is informed every time the node receives such broadcasts of its neighbors. This list contains the neighbors *ids* and their locations. Finally, each node deletes its master key, that is utilized to compute its cell key, in order to prevent an attacker from gaining any information about the key of other nodes.

Generation Phase

The generation phase is responsible for the operations that manage the detection and the verification of an event. At the time, an event is detected from one node, a confirmation message is constructed. Then the node encrypts the confirmation message and broadcasts it. An event is considered legitimate when a node receives *R* confirmation messages, concerning this event, from distinct cell nodes. Otherwise, if the node receives less than *R* confirmation messages, the node that produces the event is considered malicious and deleted from the trusted list of the other nodes. *R* is set during the implementation of the network from the administrator and the choice or *R* value leads to a trade-off between the likelihood of false positives and how strong will be the security. Using a large number of *R* offers stronger security, since the attacker must infect a large number of nodes within a cell in order to create a fake event. On the other hand, using a large number of *R* may increase the likelihood that well-behaved nodes will be considered as malicious, since the event may detected from less than *R* nodes.

When the number of received confirmation messages is equal or greater than *R,* the node constructs an event message to be forwarded to the sink. This message is digitally signed using one-way hash function and is encrypted using the private key of the node. Source *id* and event type in plaintext are also contained in the event message. Hence, it is not needed to be decrypted from the other nodes that route this message towards the sink. Consequently, the power consumption at these nodes is reduced. In addition, the node should find two paths en route to the sink. The data path for the transmission of the event message and the control path to

monitor the progress of the transmission of the event message from the one cell to the next. All the nodes that are contained in the control path are called helper nodes.

One node chooses the next hop node having in mind three conditions:

- It must exist in its trusted neighbor list.
- It must be closer to the sink.
- It must have other node in its communication range.

Forwarding Phase

The forwarding phase concerns the hop-by-hop transmission of the event message through the intermediate nodes towards the sink.

Verification Phase

The verification of an event takes place at the sink. The sink can verify whether an event is transferred by a specific node or not using the signature of the node that construct the event. Firstly, sink computes the hash code of the event and then, the received message digest (*MD*) is decrypted. There follows a comparison between those two. If they are not equal, either the signature was created without the use of the private key of the generator node, or the data was changed after signing.

Authors, also explained how *MASA* achieved the following security properties:

- Data Integrity
- Data Confidentiality
- Data Authentication
- Data Availability

Data Integrity

The signing of the valuable data is done using the private key of the generator node. Due to the fact that the corresponding public key exists only at the sink, the sink has the ability to verify that the data are not tampered and data integrity is established. Moreover, the construction of the control path for monitoring the event transmission provides hop-by-hop data integrity.

Data Confidentiality

In *MASA*, no one else except from the sink can use the appropriate key that is being utilized for the signing of the valuable data of an event. Hence, the intermediate nodes can route the event without any additional computation overhead, by using an unencrypted part of the event that exists for this reason. This ensures that only the sender and the receiver are able to see the actual content of the message and therefore, data confidentiality is achieved.

Data Authentication

In *MASA*, each node has a list of trusted neighbors. Using this list and the helper nodes towards control path, data authentication is accomplished.

Data Availability

In *MASA*, if a node operates maliciously it will be deleted from the trusted list of its neighbors. As a result, the number of the trusted nodes of the networks is reduced and network degradation is established since the compromised nodes are annihilated.

Finally, *MASA* improves some of the weakness of *LEDS* scheme, for instance the message event in *LEDS* is broadcasting in order to be transferred to the sink, compared to *MASA* that does not used broadcasting.

Conclusion and critical evaluation

In this article, the authors produce an end-to-end security scheme using a mixture of asymmetric and symmetric approaches. Nevertheless, this scheme cannot be used in *6LoWPAN* because it utilizes two paths for the data transmission, one for event message transmission and that used to control the transmission progress of the event message from the one to the next one. This cannot be done in *6LoWPAN*, since data transmissions are accomplished with respect to the routing protocol of the existing *IP*-based infrastructure. If we use this routing protocol, the control path would be removed. Therefore, data integrity and authentication will not be established any more, since they were accomplished using the control path.

3.1.7 Sizzle

Gupta *et al.* in [42] provided a small footprint *HTTPS* stack, called *Sizzle*, which is one of the smaller secure web servers in the world with respect to resource usage and physical dimensions. This is a fully implemented end-to-end security architecture for highly constrained embedded devices and contains one gateway between the monitoring station and the devices being monitored. They produced efficient implementations of public-key cryptography that can be applied in these devices and therefore Secure Sockets Layer (*SSL*) can be used. This security architecture is different from the other gateway-based architecture, such as *WAP 1.0* [43]. In their approach, all data remains encrypted along the path of the gateway, in contrast to the other gateway-based architectures where the gateway receives the data decrypted and re-encrypts it before sending it along, and hence, can see all the traffic. In addition, they proposed some protocol improvements in order to decrease the amount of data that are transferred towards the wireless hop. As a result, the latency and energy consumption an *SSL* handshake needs are reduced.

Conclusion and critical evaluation

As was stated before, *SSL* or *TLS* can only be used over *TCP*, which is not utilized in *WSNs* so often. Hence, *DTLS*, an adapted version of *TLS* for *UDP*, can be altered and used in the same way as *SSL*. Nevertheless *DTLS* is also rarely used in *WSNs*.

3.1.8 SmartTB

Weining W. and Alain Z. in [44], tried to secure the connection between each sensor and the server in "*SmartTB*" project and therefore they provided various security mechanisms:

- Security mechanisms in *MAC* sub-layer of *IEEE 802.15.4*: Provides data encryption, frame integrity, access control and sequential freshness .
- Compressed *IPsec* for *6LoWPAN*: The compressed *IPsec* produced in [5] (see page 9 of this report).
- Security mechanisms described in the *CoAP* protocol [45]: This protocol secures the *CoAP* messages using Datagram Transport Layer Security (*DTLS*) or *IPsec* (for alternative choice).

Conclusion and critical evaluation

IEEE 802.15.4 provides hop-by-hop security in contrast to *IPsec* and *DTLS* which provide end-to-end security. Therefore, in order to succeed end-to-end security in *6LoWPAN*, *DTLS*, which is an adaptation of *TLS* for *UDP,* can be used but as mentioned above, it is not widely used in *WSNs*.

3.2 End-to-end secure communications with data aggregation

End-to-end security can be combined with data aggregation if it can be provided using hop-by-hop encryption and decryption. This can be achieved in homogeneous networks with high resilient links or heterogeneous networks containing some intermediate nodes that will be used only for data aggregation and data transmission.

3.2.1 End-to-end secure communication using differentiated key pre-distribution

Gu *et. al.* in [6], provided an end-to-end secure communication protocol for *WSNs* using hop-by-hop encryption and decryption. Their approach is dependent on a methodology called "differentiated key pre-distribution". The basic idea of this technique is to reinforce the resilience of particular links by distributing different number of keys to different sensor nodes. These high resilient links are considered better to be used for data transmission, in contrast with links with low resilience. This leads to a balance between the end-to-end secure communications and the lifetime of the *WSN*, as well. In addition, they extend the location centric Greedy Perimeter Stateless Routing (*GPSR*) [46] and the data centric (minimum hop) [47] routing protocols, in order to tune them in such way that they have the link resilience as a basic metric during the routing.

Hence, this protocol is based on two components:

- Differentiated key management.
- Resilience aware routing.

Conclusion and critical evaluation

Their protocol provides the ability for data aggregation, which is important in *WSNs* for energy saving, since they utilize hop-by-hop data encryption and decryption. Nevertheless, there is a gap in this protocol. Data routing is accomplished in such a way that they will be forwarded toward a specific path. Therefore, in case a black hole (a compromise node) is maliciously inserted in this path will lead to the loss of all the transmitted data that are forwarded through this path. Moreover, this protocol cannot be used in *6LoWPAN* due to the fact that it uses a specific routing protocol, that the authors developed, in order to route the data through the high resilience links. In *6LoWPAN* this is not feasible, because it must use the routing protocol of the existing *IP*-based infrastructure. The traditional *WSNs* make the assumption that are used in homogeneous networks with *802..5.14* links. *6LoWPAN* uses a heterogeneous network with sensor nodes integrated with the Internet. Therefore the assumption that the data will be transferred through the high resilient links cannot be established and as a result, this scheme cannot be utilized in *6LoWPAN*.

3.2.2 Security framework for WSNs

Zia T. and Zomaya A. in [48], provided a security framework for *WSNs* composed of three phases :

- Cluster formation
- Secure key management
- Secure routing

Cluster formation

Each cluster contains neighbor nodes that are connected with each other. One of those nodes, in each cluster, behaves as the leader which is responsible for all the communications into its cluster and can also aggregate the packets and send them to the base station. This architecture is developed in a tree-based network topology and the data transmissions are established as follows:

- Sensor nodes collect the data and send them to the cluster leader.
- Cluster leader aggregates the data (if possible) and sends them to the next level cluster.

Eventually data will reach the base station.

Secure key management

They provided a secure hierarchical key management scheme using three keys. One of them is utilized in network generated cluster key to address the hierarchy in sensor network, and the other two pre-deployed keys are used in all nodes.

Secure routing

- They developed the two following algorithms in order to achieve secure data transmission between the nodes and the base station:
  - Sensor node algorithm:
    - Sensor nodes encrypt and transmit the data using K_n (network key).
    - The encrypted data are transmitted from nodes to cluster leader.
    - Cluster leader adds the *ID* number to the data and sends them to the higher level of cluster leaders. In order to decrypt the data, cluster leader uses K_c (cluster key) and encrypts them utilizing K_n.
  - Base station algorithm:
    - Broadcast K_s (sensor key) and K_n by the base station.
    - Decrypts and authenticates the data by the base station.

Conclusion and critical evaluation

This security framework provides the ability for data aggregation as well but it cannot be used in *6LoWPAN* networks. First of all *6LoWPAN* network cannot be split into clusters since the whole Internet is part of the network and also routing algorithm cannot be adapted, for the reasons explained above.

3.2.3 CDAP

Ozdemir S. in [49] presented a protocol that takes advantage of privacy homomorphic cryptography, which allows direct computations on encrypted data in order to achieve both secure communication and data aggregation, in *WSNs*. This protocol is called Concealed Data Aggregation using Privacy Homomorphism (*CDAP*). He assumed a static cluster based *WSN* illustrated in Figure 16 that contains a lot of sensor nodes, some aggregator nodes (*AGGNODEs*) and a base station. Sensor nodes are nodes with low power capabilities and *AGGNODEs* are more powerful sensor nodes with more memory space. *AGGNODEs* collect the encrypted data from their neighboring sensors and aggregate them. Then, the encrypted data are forwarded to the next level cluster and eventually reach the base station. Data encryption and aggregation are made using using privacy homomorphism. Their evaluations show that this scheme is viable with large heterogeneous *WSNs*.

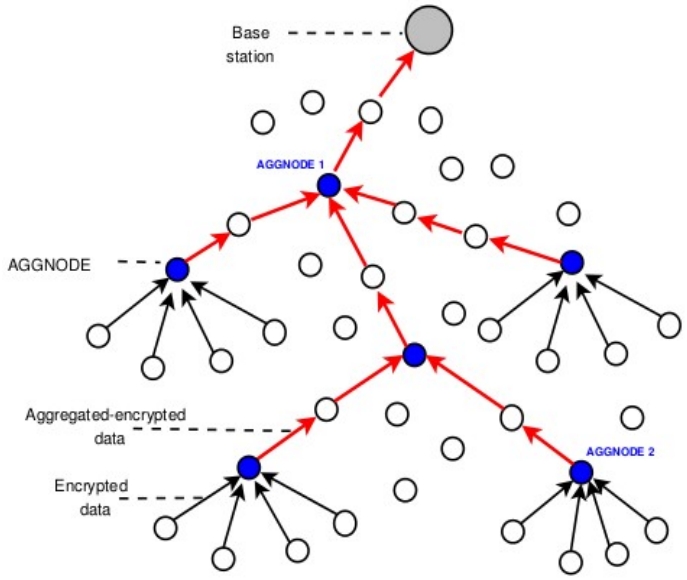


Figure 16: The aggregation scenario of Concealed Data Aggregation using Privacy Homomorphism (CDAP). AGGNODEs collect the ecrypted data from their neighboring sensors, aggregate them and forward them to the next level cluster. (From [49].)

Conclusion and critical evaluation

The author presented a protocol that provides end-to-end secure communication in large heterogeneous *WSNs*. However, it is not feasible for *6LoWPAN* due to the fact that it works only for the specific static cluster based *WSN,* which differs from *6LoWPAN* infrastructure explained in section three.

3.2.4 Secure scheme for a multilevel network

Castelluccia *et al.* in [50], provided a simple and secure encryption scheme which allows data aggregation in *WSNs*. They considered a multilevel network tree with multiple sensor nodes and a sink. Moreover, data encryption is done using homomorphic encryption. Therefore, when the aggregator receives the encrypted data, it is able to aggregate them without the need to decrypt them first. Furthermore, in addition to the ciphertexts, information of non-responding nodes are also forwarded that are used by the sink in order to decrypt the data. As a result, there is an additional overhead, since the number of bits transmitted increases.

The evaluation of the performance of their scheme shows that it is hardly less bandwidth-efficient than the hop-by-hop scheme, but it produces a stronger level of privacy than the end-to-end encryption scheme without aggregation.

Conclusion and critical evaluation

This encryption scheme provides both end-to-end security and data aggregation due to the fact that it utilized homomorphic encryption which allows arithmetic operations on ciphertexts. Moreover, the multilevel network tree contains some aggregator nodes in order to aggregate the data. This scheme, can be probably used in *6LoWPAN* with or without a few adaptations, since *6LoWPAN* is tree-based network.

3.2.5 SEEDA

Poornima A. S. and Amberker B. B. in [51], introduced a secure data aggregation scheme which provides end-to-end data privacy in *WSNs*, called *SEEDA* (Secure End-to-End Data Aggregation in Wireless Sensor Networks). They considered that a *WSN* consists of multiple sensing nodes (*SN-nodes*), some aggregator nodes (*AG-nodes*) and a sink. *SN-nodes* sense the data and *AG-nodes* aggregate the data and forward them to their parents. Moreover, the network is organised as a *m-ary* tree.

Data encryption is established at *SN-nodes* using homomorphic encryption, which allows aggregation on ciphertexts, and data decryption is achieved at the sink. At *AG-nodes* the ciphertexts are added. In addition, a new ciphertext with regard to the non responding nodes is created, considering the data to be *0*. This ciphertext is forwarded along with the encrypted data. As a result, the sink can obtain the aggregated data using the analogous subtraction on these ciphertexts. Therefore the number of bits transmitted are reduced, since no additional information about the non responding nodes are sent.

This scheme is a hybrid scheme which takes advantage of the best features of end-to-end and hop-by-hop aggregation scheme. End-to-end data privacy is established as in any end-to-end aggregation scheme and the number of transmission bits is almost the same with the number of transmission bits in a Hop-by-Hop aggregation scheme.

They evaluate their implementation by comparing *SEEDA* with hop-by-hop, end-to-end and All-node and the results show that the number of bits that are transmitted toward the tree in *SEEDA* are reduced from 30% - 50% comparing to the end-to-end aggregation scheme described in [50].

Conclusion and critical evaluation

This scheme also provides end-to-end security using homomorphic encryption and therefore data aggregation can be accomplished. Moreover, it can probably be used in *6LoWPAN* with or without a few adaptations, since *6LoWPAN* is tree-based network, and it's worth investigating.

4. Project's context and its relation with IPv6 Security in the IoT

4.1 Project's context

As stated above, this project will conduct an in-depth evaluation of the performance of *IPsec* in *6LoWPAN* networks of resource-constrained embedded devices. *IPsec* is mandatory for the *IPv6* and therefore, it would be beneficial if *IPsec* could be efficiently utilized in *6LoWPAN*. This is because all the existing endpoints on the Internet would not need to be modified in order to communicate securely with the *WSN*. In addition, there is no need for a trustworthy gateway using *IPsec,* because true end-to-end security is implemented. According to the literature review, regarding the security in *6LoWPAN*, two implementations of *IPsec* for *6LoWPAN* have been developed. Thus, the *IPsec* implementation that is described in [5] was selected to be extended and evaluated, because the given results illustrate that it can be efficiently used in *6LoWPAN*. This implementation is incompatible with the Routing Protocol for Low-Power and Lossy Networks (*RPL*) [52]. *Contiki OS* utilizes *RPL* and thus *IPsec* implementation is also not compatible with the *Contiki OS*. Therefore, it had to be extended and modified in order to work. More details regarding the reason why it did not work with the new version, the changes made and why it was important to work with the newest version of the *Contiki OS,* are described in the next chapter in section 5.1. Moreover, many experiments have been performed in order to investigate the energy consumption overheads, the end-to-end delay, the memory footprints and the packet loss using *IPsec*.

Furthermore, during my research a lightweight implementation of *DTLS* for the *Contiki OS* has been found, that is tested on Raven boards. However, Raven boards have more memory capabilities than Tmote sky boards and therefore, an examination if it is memory permissible on Tmote sky has been accomplished. Consequently, memory footprints of this implementation have been investigated. As stated above, it is a benefit if sensor nodes are able to communicate with Internet host utilizing security solutions that can be combined with the ones that already exist in the Internet. Considering that altering the Internet is very difficult, for instance by proposing new protocols, it is vital for sensor networks to adopt security solutions already developed in the Internet. *DTLS* is widely spread into the Internet and thus it is crucial for *WSNs* to be able to adopt this protocol.

Additionally memory footprint comparisons between the extended *IPsec* implementation presented here, the already existing and the optimized *DTLS* implementation have been made. Finally a general comparison of *IPsec* and *DTLS* in the area of the *IoT* has been accomplished.

4.2 Project's relation with background

This project was based on the results of previous studies and builds upon their results. Because *6LoWPAN* has been recently demonstrated there is still lack of research regarding its end-to-end security. Therefore, the background literature review extended beyond the *6LoWPAN* research effort, into traditional *WSNs.* This was done in order to decide if an end-to-end security solution utilized in the latter can be efficiently used in *6LoWPAN* as well.

Part of the project's aim, was to investigate how the *Contiki OS* and the *Cooja* simulator work. Hence, some experiments using the already existing examples in the *Contiki OS* have been executed, before the investigation of the selected methods.

Since this project will evaluate the performance of *IPsec* in *6LoWPAN*, a further study regarding the *IPsec* and its components (e.g *AH, ESP*) was required. Moreover, a more in-depth research about *TLS*/*DTLS* was needed as well, in order to be able to investigate if *DTLS* can be loaded on Tmote sky boards and to finally accomplish a comparison of *IPsec* and *DTLS.*

5. IPsec

5.1 Analysis of the Contiki IPsec implementation

As mentioned above, the *IPsec* implementation provided in [5] was chosen to be extended and evaluated. This implementation has been extensively analysed in chapter 3. As outlined in this section, this implementation supports both *IPsec's AH*, that provides data integrity and authentication, and *ESP*, that provides confidentiality, integrity and authentication. Moreover, only transport mode has been developed, since tunnel mode in the context of *6LoWPAN* seems not practical, as the packet size would be further increased due to the additional headers.

More specifically, for the purpose of this implementation, the existing *Contiki µIP* stack, which already produces *6LoWPAN* functionality, needed to be modified. This is utilized on sensor nodes and on a soft bridge that connects the *WSN* and Internet. *IPsec/6LoWPAN* compression mechanisms are developed, as described in section 3.3.1.1, in addition to the *IPsec* protocol. Furthermore, *NHC_EH*, *NHC_AH* and *NHC_ESP* encodings (see section 3.3.1.1) are supported at the *SICSLoWPAN* layer, which is the *6LoWPAN* component of the *µIP* stack.

In addition, they developed all the cryptographic modes of operation required for authentication and encryption in *IPsec*, and they utilized *SHA1* and *AES* implementations from *MIRACL* [53], an open source library. They implemented the *AES-XCBC-MAC-96* and *HMAC-SHA1-96* for *AH*, and *AES-CBC* and *AES-CTR* for encryption, and *AES-XCBC-MAC-96* and *HMAC-SHA1-96* for authentication, for *ESP*. *SAs* are achieved using pre-shared keys, and since a pre-shared mechanism is mandatory in *IPsec*, this implementation works with any IPv6 host on Internet.

However, this implementation has one essential limitation in order to work correctly. The *AH* and *ESP* header must be the first extension header following the *IP* header in the *IP* datagram, when using *IPsec* with *AH* and *IPsec* with *ESP* respectively, as shown in Figure 17. This is a crucial restriction, since the *IP* datagram should always sustain this format, no other *IPv6* extension header can be used. Hence, even though all sensors can be set to operate under this restriction, it is not feasible and viable to be adopted from all the endpoints of the Internet. Therefore, this implementation is problematic and only works under the assumption that the *IP* datagrams do not contain any other *IPv6* extension headers except of the *AH* or *ESP* header.

IP header	AH header	Payload (e.g TCP, UDP, ICMP)

a) IP datagram using IPsec with Authentication Header

IP header	ESP header	Payload (e.g TCP, UDP, ICMP)	ESP trailer	ESP auth

b) IP datagram using IPsec with Encapsulation Security Payload

Figure 17: IP datagram for the IPsec implementation in 6LoWPAN. AH and ESP header must be the first extension header after IP header accordingly.

Another problem of this implementation, found during the examination, is that the validation of the *MAC* for both the *AH* and the *ESP* header is not implemented correctly. The validation is accomplished as shown in Figures 18 and 19.

```
if (memcmp(ah_header->mac, mac, IPSEC_MACSIZE) > 0 ) {
      PRINTF("IPsec-AH: MAC is wrong, will drop current packet\n");
      goto drop;
}
```

Figure 18: Validation of *MAC* for Authentication Header

```
if (memcmp(esp_header->data + encrypted_data_len, mac,IPSEC_MACSIZE) > 0){
   PRINTF("IPsec-ESP: MAC is wrong, will drop current packet\n");
   goto drop;
}
```

Figure 19: Validation of *MAC* for Encapsulating Security Protocol

The function *memcmp* performs a bitwise comparison between two memory blocks and returns an integer greater than, equal to or less than *0* if the first string is greater than, equal to, or less than the second string respectively[3]. In *IPsec* implementation the *MAC* of the transferred packet is compared with the new *MAC* calculated for this packet. Therefore, the *MAC* is valid only if the two *MACs* are equal. Hence, the validation should be done by checking if the returning value of *memcmp* is equal (=) to *0* and not greater (>) than *0* as it is done in this implementation and illustrated in Figures 18 and 19.

Consequently, the results provided from the authors regarding the evaluation of this implementation may not be so accurate, since packets with not a valid *MAC* may be considered valid and this affect the final results about the performance of *IPsec* in *6LoWPAN*.

5.2 Implementation

As found from the analysis of the existing implementation, an essential constrain exists, that cannot be omitted, in order to work correctly. During the examination of this implementation I had to deal with this limitation when I attempted to incorporate it into the *Contiki OS* version 2.6. This version of the *Contiki*, the *RPL* [52] has been established as the *de facto* standard for routing in *6LoWPAN*, and *RPL* header should be the first extension header after *IP* header in *IP* datagram. Therefore, this implementation does not work any more, not even in *6LoWPAN*. Consequently, it was necessary to be modified and extended in order to work correctly according to the new requirements. Hence, the appropriate modifications in the *Contiki µIP* stack have been made and this extended implementation not only works using the *RPL* protocol, but it also works correctly if any other *IPv6* extension header exists in the *IP* datagram.

Next extension headers of an incoming message are processed in the following order:

1. *UDP* header: If the next header is a *UDP* header, it means that no other extension headers exist in the packet. Thus, it continues with the processing of the other headers of the packet.

2. *ICMP6* header: If the next header is an *ICMP6* header it means that there are no other extension headers in the packet. Hence, it continues with the processing of the other headers of the packet.

3. *HBHO* header: If the next header is a *HBHO* header it process the header. If the header is not valid the packet is dropped. Otherwise, the *HBHO* header is removed and starts from the beginning to check the next header.

4. *ESP* header: If the next header is an *ESP* header, the header is processed and *MAC* validation is performed. If the *MAC* is not valid, the packet is dropped. Otherwise, the *ESP* header is removed and the process continues to check the next header, starting from the beginning of this process.

3 From <http://linux.die.net/man/3/memcmp>.

5. *AH* header: If the next header is an *AH* header, the header is processed and *MAC* validation is performed. If the *MAC* is not valid the packet is dropped. Otherwise the *AH* header is removed and the process continues to check the next header, starting from the beginning of this process.

6. Destination Option Header: If the next header is a Destination Option header, the header is processed. If the header is valid, it is removed and the process continues to check the next header, starting from the beginning of this process. If it is not valid the packet is dropped.

7. Routing Header: If the next header is a Routing header it processes the header. If the header is not valid the packet is dropped, otherwise the Routing header is removed and the process continues to check the next header, starting from the beginning of this process.

8. Fragment Header: If the next header is a Fragment header it processes the header. If the header is valid the Fragment header is removed and the process continues to check the next header, starting from the beginning of this process. If the header is not valid the packet is dropped.

9. NONE: If the next header is equal to NONE it means that there is no other header in the packet. Thus the packet is corrupted and it will be dropped.

This process is performed until all the extension headers will be processed and removed from the packet and it is illustrated in Figure 20.

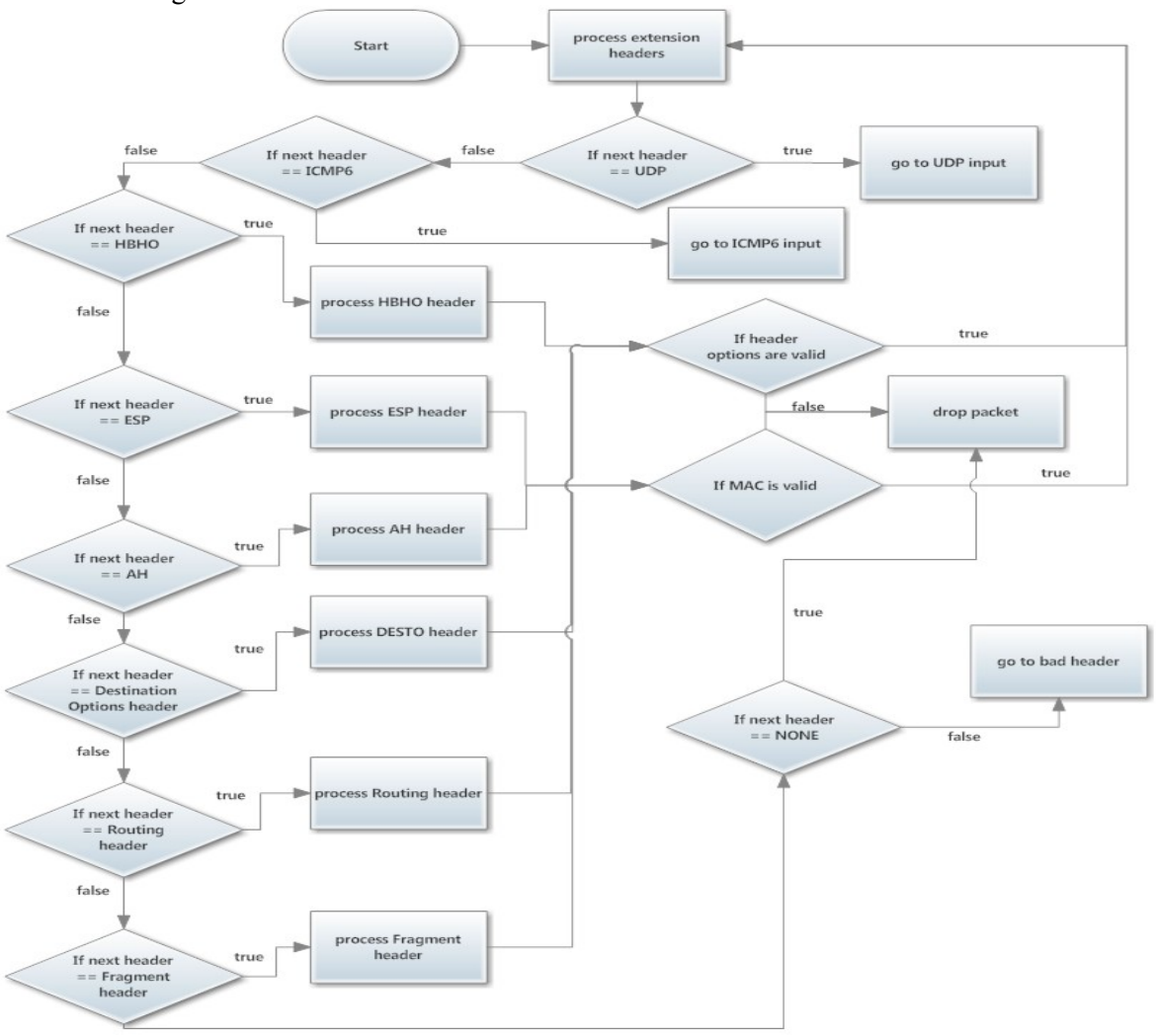


Figure 20: Flowchart of the extension header processing in an incoming packet.

Figure 21 shows how the extension headers are processed in an output packet. Firstly, a check if any extension headers exists in the packet is performed. If such a header exists, while the next header is neither *ICMP6* nor *UDP* or *TCP*, next header is omitted and its length is added to a local variable that stores the extension header's length.

```
if(uip_ext_len > 0) {
   while (((struct uip_ext_hdr *)locnexthdr)->next != UIP_PROTO_ICMP6 &&
          ((struct uip_ext_hdr *)locnexthdr)->next != UIP_PROTO_UDP &&
          ((struct uip_ext_hdr *)locnexthdr)->next != UIP_PROTO_TCP) {
               locnexthdr += ((struct uip_ext_hdr *)locnexthdr)->len +
                             ((struct uip_ext_hdr *)locnexthdr)->len *
                             ((struct uip_ext_hdr *)locnexthdr)->len;
   }
   next_header = ((struct uip_ext_hdr *)locnexthdr)->next;
 }
```

Figure 21: Extension headers processing in an output packet. The headers are omitted until the next header is ICMP6 or UDP or TCP.

Figures 22 and 23 illustrate how *AH* and *ESP* header are added to an output packet after the extension headers processing described above, accordingly. If any extension header exists in the packet, the *AH* or *ESP* header is added after the last extension header. Otherwise, is added after the *IP* header. Using the *memmove* method the *IP* payload is moved, as much as needed, leaving space for the *AH* and *ESP* header respectively.

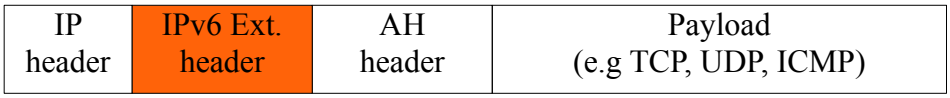
```
/* Backup next header before updating to "AH" */
if(uip_ext_len > 0) {
     ((struct uip_ext_hdr *)locnexthdr)->next = UIP_PROTO_AH;
} else {
     UIP_IP_BUF->proto = UIP_PROTO_AH;
}
/* Move IP payload, leaving space to AH header */
memmove(((char *) UIP_AH_BUF) + sizeof(struct uip_ah_header),
         UIP_AH_BUF, uip_len - sizeof(struct uip_ah_header) -
               uip_l2_l3_hdr_len);
```

Figure 22: Authentication header adding in an output packet. AH is added after the last extension header, if any extension headers exist. Otherwise, it is added after the IP header.

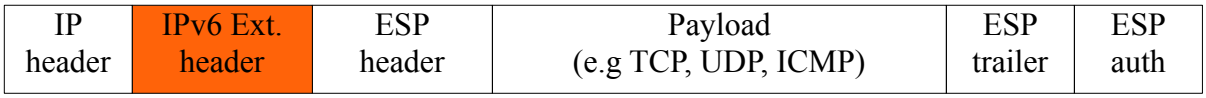
```
/* Backup next header before updating to "ESP" */
if(uip_ext_len > 0) {
     ((struct uip_ext_hdr *)locnexthdr)->next = UIP_PROTO_ESP;
} else {
     UIP_IP_BUF->proto = UIP_PROTO_ESP;
}
/* Move IP payload, leaving space to ESP header */
memmove(((char *) UIP_ESP_BUF) + sizeof(struct uip_esp_header),
         UIP_ESP_BUF, encrypted_data_len);
```

Figure 23: Encapsulating Security Payload header adding in an output packet. ESP is added after the last extension header, if any extension headers exist. Otherwise, it is added after the IP header.

Figure 24 illustrates the *IP* datagram needed for the extended *IPsec* implementation presented here. Since this implementation, can handle *IP* datagrams composed of any *IPv6* extension header, sensor nodes in *6LoWPAN* will be able to communicate with any other *IPv6* host on Internet, without the need of any modification at the host. Moreover, I amended the way the validation of the *MAC* is established. During the implementation, Wireshark [54], a network protocol analyzer, was used as a means to examine if *IPsec* works correctly. Using the *pcap* (packet capture) feature of the Wireshark, the packets of the network were captured and their protocols were analyzed. Therefore, it was easy to understand the headers the packets consist of and their order in the *IP* packet.



a) IP datagram using IPsec with Authentication Header



b) IP datagram using IPsec with Encapsulation Security Payload

Figure 24: IP datagram for mine extended IPsec implementation in 6LoWPAN. Any IPv6 extension header could exist after the IP header.

In order to quantify the performance of this implementation many experiments have been executed, in which two different networks composed of a different number of sensors, and packets with different data sizes (8, 16, 32 and 64 bytes) were used. Moreover, an *AES* implementation was used provided by Texas Instruments (*TI*) [55], in addition to the *AES* implementation provided by *MIRACL*, comparing them in terms of end-to-end delay. Furthermore, the *AES-XCBC-MAC-96* for *AH* and the *AES-CTR* for encryption were used and *AES-XCBC-MAC-96* for authentication for *ESP*. The implementation was evaluated in terms of energy consumption, delay, packet loss and memory footprint under different configurations. More details regarding the experimental setup and the results are provided in the next chapter.

6. IPsec Evaluation

6.1 Introduction

All the experiments have been performed using *Cooja* simulator and a *PC* running *Ubuntu 11.10 OS* with *IPsec* enabled.

In order to be able to execute these experiments, the *IPsec* was reconfigure in the above *PC*. This was achieved, following the instructions of [56] and by altering the ipsec-tools.conf file provided by the authors of [5] in order to set mote *IPv6* addresses. Moreover, Wireshark was used in order to examine the correctness of this implementation before its performance evaluation, as outlined above.

For the performance evaluation, two different networks were used:

- Network 1:
  - 1 Linux machine running *Ubuntu 11.10* with *IPsec* enabled
  - 1 *6LoWPAN* soft bridge (implemented by one Tmote sky)
  - 6 Tmote sky sensor nodes

- Network 2:
  - 1 Linux machine running *Ubuntu 11.10* with *IPsec* enabled
  - 1 *6LoWPAN* soft bridge (implemented by one Tmote sky)
  - 13 Tmote sky sensor nodes

Figure 25 describes the experimental setup and how the Linux machine communicates with the simulator. The Tmote sky sensor nodes and the *6LoWPAN* router correspond to the simulated environment. Linux machine communicates with the simulated environment over a virtual loopback interface. More specifically, *Cooja* simulator connects to the Linux machine via one socket and listens for packet. Any packet that is sent to the loopback from the Linux machine is immediately forwarded to the serial interface of the *6LoWPAN* router. Afterwards, *6LoWPAN* router forwards the packets to the appropriate sensor node.

The Tmore sky sensors, execute a simple echo server that listens to a fixed *UDP* port. At the time a packet is received, *6LoWPAN* layer processes it, and it is interpreted by the *IPsec* layer and by the *μIP*. Afterwards, its payload is transferred to the application. As a reply, a new datagram of the same size is constructed and is sent back using the opposite process. Therefore, the end-to-end communication between the sensor node and the Internet host is secured using *IPsec*.

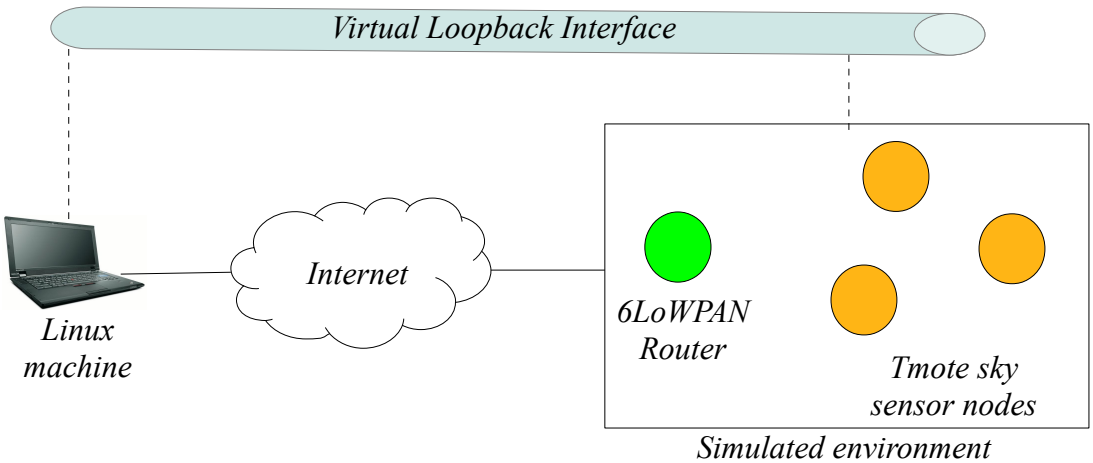


Figure 25: Experimental Setup. Linux machine communicates with the simulated environment over a Virtual Loopback Interface.

The parameters used in the experiments are outlined in Table 3. *Contiki's CSMA MAC* and *Contikis's ContikiMac* drivers were utilized. This is because these drivers allow packet retransmission after a collision detection event and provide very good power efficiency by keeping the radio off as much as possible and periodically check for radio activity in the radio medium respectively.

Unless otherwise stated *100* packets were used for each experiment in order to get more accurate results. The data size of each packet ranges form *8* to *64* bytes and the exact number is given in each of the experiments below. Moreover, two embedded software implementations of *AES* exist, namely *MIRACL AES* and *TI AES.* The performance of these implementations was evaluated in terms of packet loss and response time. The purpose of this was to choose one of these two algorithms in order to evaluate the performance of *IPsec*. The results are presented in section 6.1.3.2 and show that the packet loss is almost the same in both implementations, whereas *MIRACL AES* is faster than *TI AES*. Thus, *MIRACL AES* was utilized in all the experiments, except the ones run for comparison performance with the *TI AES* implementation. In addition *AES-XCBC-MAC-96* mode of operation was utilized for *AH,* and *AES-CTR* and *AES-XCBC-MAC-96* for encryption and authentication in *ESP*, accordingly.

Nodes in Network 1	7 Tmote sky sensor nodes (1 soft bridge, 6 simple echo servers)
Nodes in Network 2	14 Tmote sky sensor nodes (1 soft bridge, 13 simple echo servers)
PC	Linux machine running Ubuntu 11.10 with IPsec enabled
MAC Layer	IEEE 802.15.4
MAC Driver	CSMA
Duty Cycling	ContikiMAC
Messages	100
Duration	5 minutes (real time inside Cooja)
Message Size	Ranges from 8 to 64 bytes
AES implementation	MIRACL and TI
AES mode of operation for AH	AES-XCBC-MAC-96
AES mode of operation for ESP	AES-CTR and AES-XCBC-MAC-96

Table 3: Simulation Configuration

The metrics that were considered during the evaluation are the following:

1. Energy consumption: Using *Contiki's* integrated energy estimation mechanisms the energy consumption of the network was measured. More specifically, energy consumption of the microcontroller (*MCU*), the radio transmitting (*TX*) and the radio receiving (*RX*) were measured, using the energest mechanism provided from *Contiki*.

2. Delay: The response time, was measured in two different ways:
    - System-wide response time: The response time is the time needed to send a packet from the Linux machine to a simulated sensor and to receive a response. This way of measuring the response time is inaccurate due to the fact that the computed time belongs to two different spacetimes; a) the time needed for transmitting the message from Linux machine to *6LoWPAN* soft bridge in *Cooja* simulator and backward, and b) the time needed for the message to be transferred from the *6LoWPAN* soft bridge to the sensor node inside *Cooja* and backward. In addition, the time needed in *Cooja* simulator is much less than the real

time that would be needed because the process in the simulator is faster than the real environment. Nevertheless, according to [5] this is an acceptable practice and it illustrates the relative delay.

- *Cooja* response time: The response time is the time needed for a packet to be forwarded from the *6LoWPAN* soft bridge to a sensor and to receive a response. This time is measured using real clock, i.e sensors' clock, instead of the linux clock that is used in order to measure the system-wide response time. In this way the real time inside *Cooja* simulator was measured.

3. Packet loss: As stated above, for each experiment *100* packets are used. Hence, packet loss is estimated by measuring for how many of them the Linux machine did not receive a response.

4. Memory footprints: The *RAM* and *ROM* footprint of this extended *IPsec* implementation, were measured by running *Contiki* in the *MSPSim* emulator [57]. *RAM* is computed as the sum of the runtime stack usage and the global data.

6.2 Estimation of the correctness of IPsec implementation

Figures 26 and 27 illustrate an analysis of a packet, which have been captured using the *pcap* feature of the Wireshark. More specifically, they show the order of the headers in a *UDP* message which has been send from the *PC* (Linux machine) to a sensor node in *Cooja* simulator, using *IPsec AH* and *IPsec ESP* accordingly.

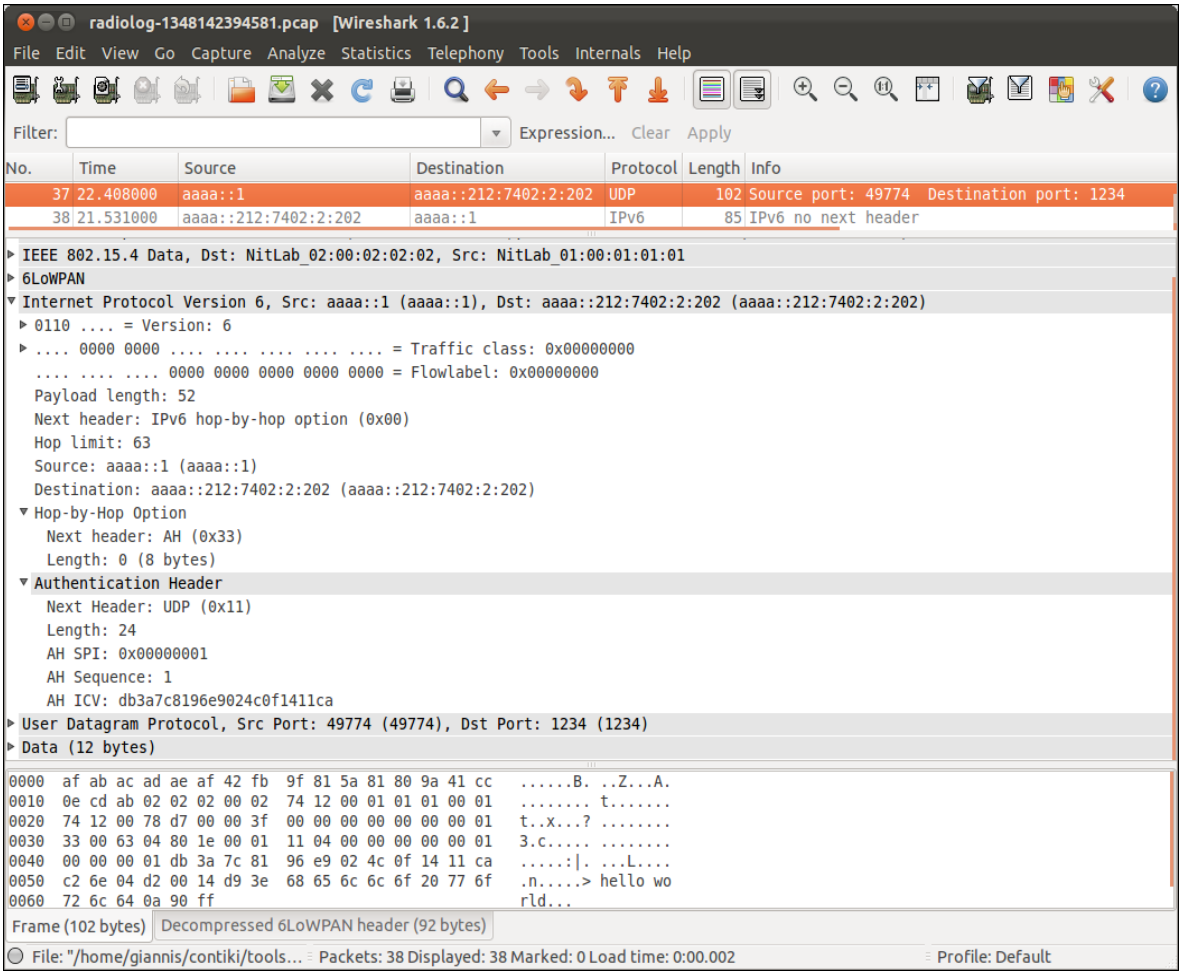


Figure 26: Packet capture using Wireshark. Analysis of a UDP packet on 802.15.4 LoWPAN using IPsec with Authentication Header.

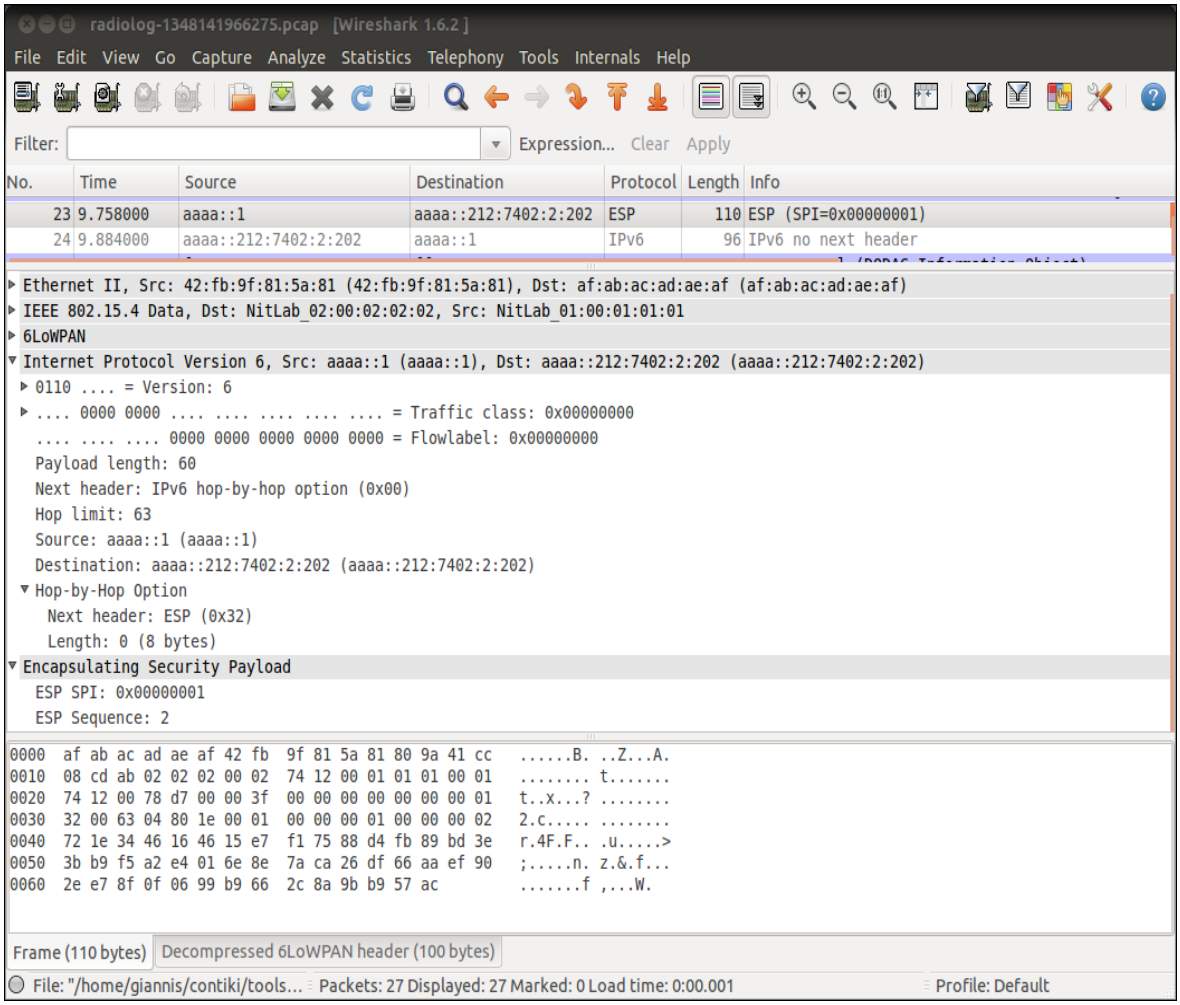


Figure 27: Packet capture using Wireshark. Analysis of a UDP packet on 802.15.4 LoWPAN using IPsec with Encapsulation Security Payload Header.

As it can be observed, the *IP* packet is composed of the following headers:

- IEEE 802.15.4 : It is the standard which determines the physical and the link layer of *LoWPANs* and exists because this is a *LoWPAN* packet.

- 6LoWPAN: Corresponds to the adaptation layer. This is the compressed *v6* header used in 802.15.4 network.

- Internet Protocol version 6: Corresponds to the network layer.

    - Hop-by-hop option: This is the mandatory *RPL* header which should certainly be the first header after the *v6* header.

    - Authentication Header: This is the *AH* header and it can be observed that it is the second header, after the *v6* and the *RPL* header.

    - Encapsulating Security Payload: This is the *ESP* header and it can be seen that it is the second header, after the *v6* and the *RPL* header.

Consequently, it can be seen that the *IP* packet follows the protocol order described in Figure 24, i.e *IP* header, Hop-by-hop Options extension header (*HBHO*), *AH, UDP* for *IPsec AH* and *IP* header, *HBHO*, ESP*, UDP* for *IPsec ESP*. Hence, *IPsec* implementation is working correctly. Moreover, Wireshark provide very good support for 802.15.4 / *6LoWPAN* and it was very helpful for the implementation and the debugging of the extended *IPsec*.

6.3 Network 1

6.3.1 Experimental Setup

The first network as described above, consists of one Linux machine, running Ubuntu *11.10*, *6* Tmote sky sensor nodes and one *6LoWPAN* soft bridge that is implemented by another one Tmote sky sensor node and is illustrated in Figure 28.

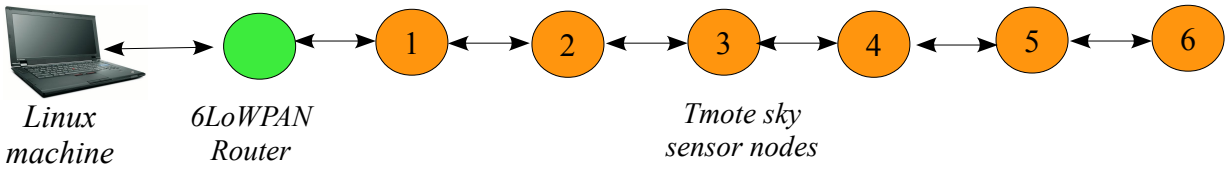


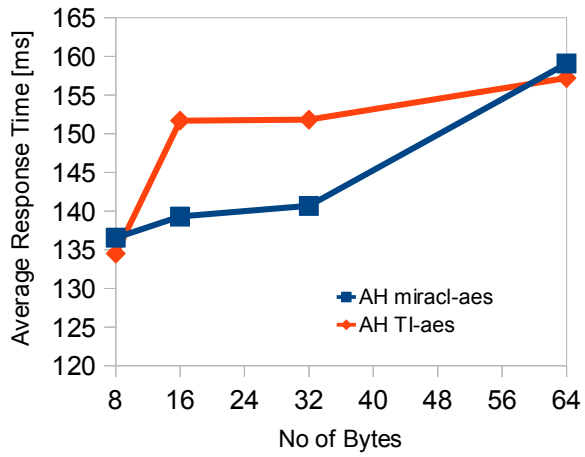
Figure 28: Network 1, composed of one Linux machine running Ubuntu 11.10, one 6LoWPAN soft bridge (implemented by one Tmote sky) and 6 Tmote sky sensor nodes.

6.3.2 Comparison of MIRACL AES and Texas Instruments AES

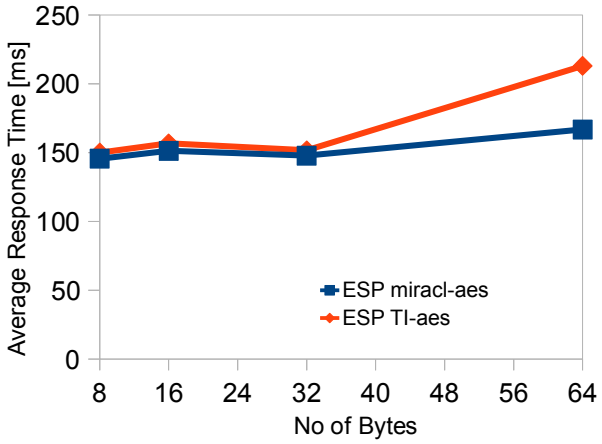
The system-wide response time and the number of packet loss were measured and evaluated for different data sizes using *MIRACl-AES* implementation and *TI-AES* implementation. Experiments were conducted using a routing distance in the *WSN* of one hop.

	Packet loss using IPsec AH		*Packet loss using IPsec ESP*	
No of bytes	*MIRACL-AES*	*TI-AES*	*MIRACL-AES*	*TI-AES*
8	5	2	0	0
16	3	5	0	0
32	3	3	8	8
64	2	2	0	0

Table 4: Packet loss for MIRACL and TI AES using IPsec AH and IPsec ESP, and different IP datagram sizes. MIRACL and TI implementations have the same number of packet loss when ESP is used while the difference is negligible when AH is utilized.



(a) Single hop with different data sizes using IPsec AH



(a) Single hop with different data sizes using IPsec ESP

Figure 29: Average Response time versus IP datagram size with MIRACL and TI AES for IPsec AH and IPsec ESP. MIRACL-AES is slightly faster than TI-AES.

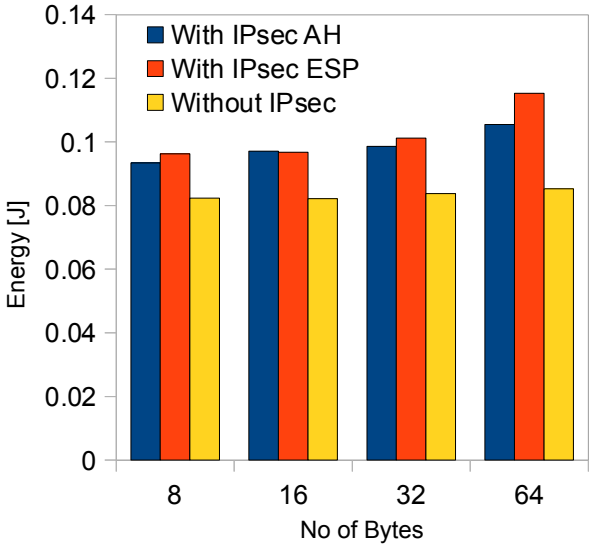
Figures 29a and 29b show the average response time using *IPsec AH* and *IPsec ESP*, accordingly, for both *MIRACL-AES* implementation and *IT-AES* implementation, in dependency of the *IP* datagram size. As it can be observed, *TI-AES* is slightly faster than *MIRACL-AES* for sizes *8* and *64* bytes, while *MIRACL-AES* is much more faster for sizes *16* and *32* bytes, when *AH* is used. In addition *MIRACL-AES* is slightly faster than *TI-AES* when *ESP* is used for all the *IP* datagram sizes. Moreover, as it can be seen from Table 4, *AES* implementation does not affect the number of packet loss, since the amount of packet loss is exactly the same for the two algorithms when *ESP* is utilized and is hardly different for *AH*.

6.3.3 Energy Consumption

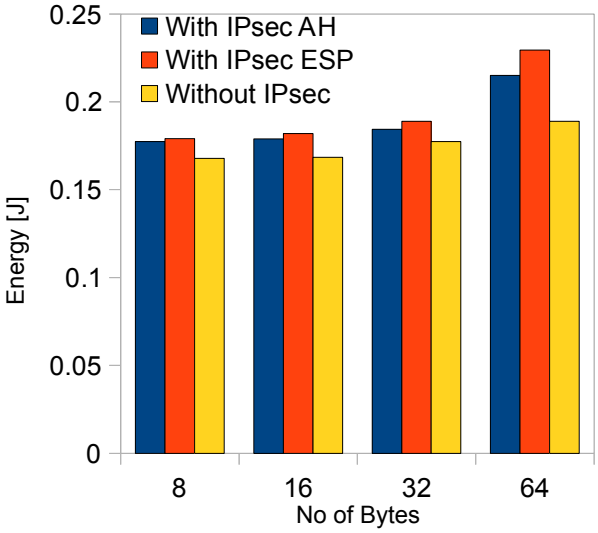
Energy Consumption using 1 hop and different data sizes

The energy consumption when using *IPsec AH*, *IPsec ESP* and without *IPsec*, for the *MCU*, the *TX* and the *RX* of the first sensor node of the network shown in Figure 28 was measured. Each experiment was run for five minutes and the messages were sent to the first sensor node.

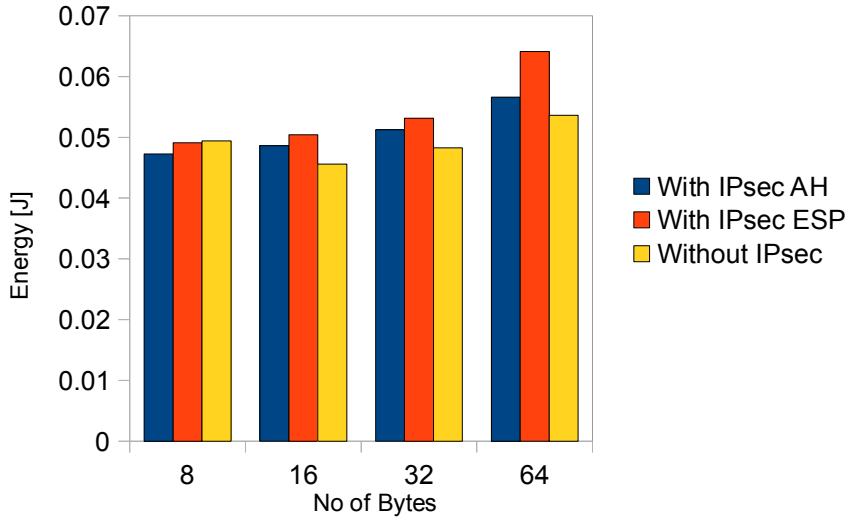
Figure 30 *a*, *b* and *c* show the energy consumption of the *MCU*, the *RX* and the *TX* accordingly. The results show that *ESP* consumes more energy than *AH* and this is because *AH* uses only authentication compared to *ESP* that uses both authentication and encryption. Moreover, energy consumption without *IPsec* is lower than with *IPsec*. Furthermore, as the datagram size grows, the energy consumption increases as well. In the worst measured case, *IPsec ESP* on *64* bytes, the energy consumption is around *0.12*, *0.22* and *0.65 Joule* for *MCU*, *RX* and *TX* respectively.



(a) Microcontroller energy consumption with different data sizes



(b) Radio receiving energy consumption with different data sizes



(c) Radio transmitting energy consumption with different data sizes

Figure 30: Energy consumption of the microcontroller, the radio receiving and the radio transmitting of the first sensor of the network when using IPsec AH and ESP and without IPsec. ESP consumes more energy than AH and the energy consumption without IPsec is lower than with IPsec.

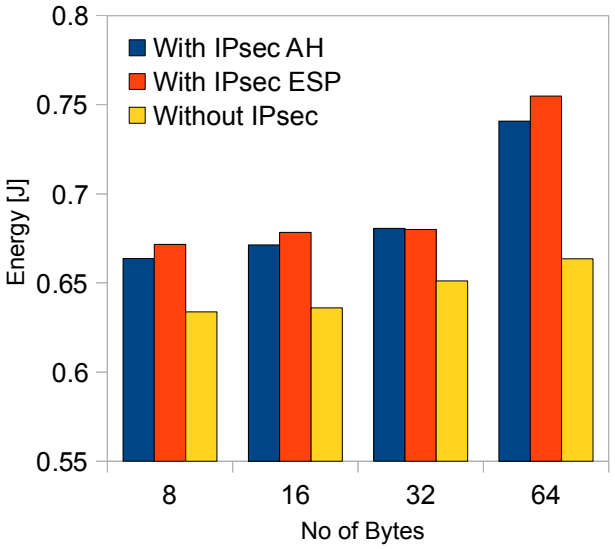
System-wide Energy Consumption

Energy Consumption using 6 hops and different data sizes

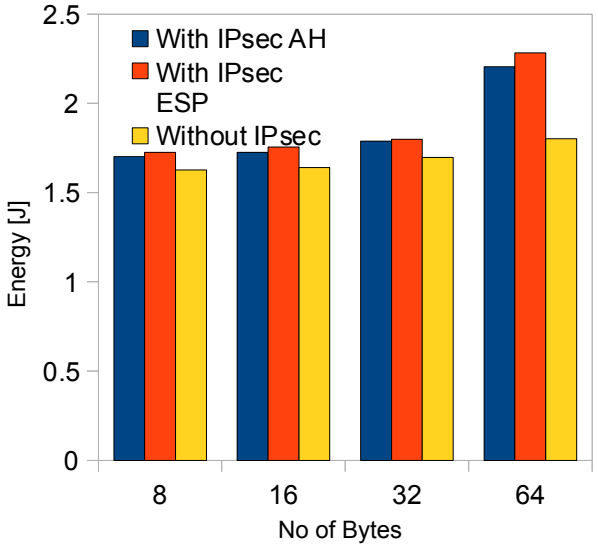
The overall energy consumption of the entire network illustrated in Figure 28, when using *IPsec AH*, *IPsec ESP* and without *IPsec* was measured, for the *MCU*, the *TX* and the *RX*. Each experiment was run for five minutes and the messages were sent to the sixth sensor node.

Figure 31 *a*, *b* and *c* show the energy consumption of the *MCU*, the *RX* and the *TX* accordingly. The results

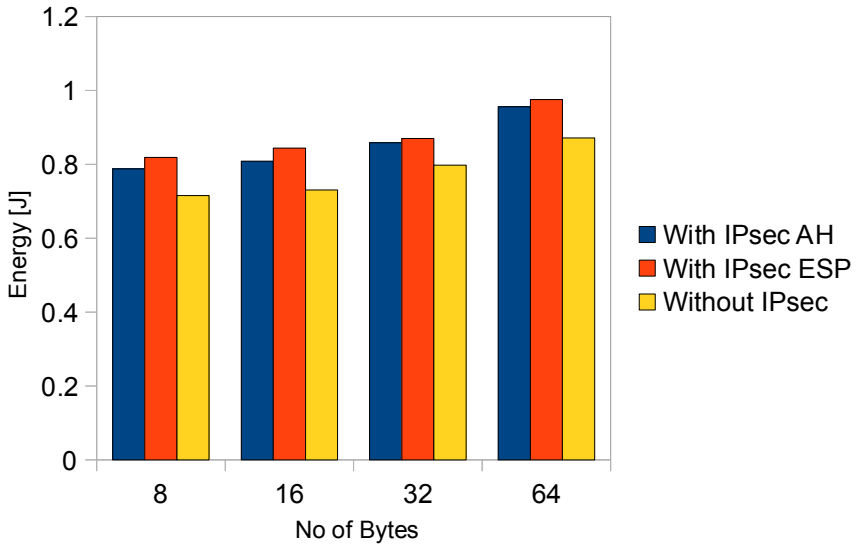
show that *ESP* consumes more energy than *AH* and this is because *AH* uses only authentication whereas *ESP* uses both authentication and encryption. Moreover, energy consumption without *IPsec* is lower than with *IPsec*. In addition, the energy consumption is increased, as the datagram size grows. In the worst measured case, *IPsec ESP* on *64* bytes, the energy consumption is around *0.75*, *0.23* and *0.98 Joule* for *MCU*, *RX* and *TX* respectively.



(a) Microcontroller energy consumption with different data sizes



(b) Radio receiving energy consumption with different data sizes



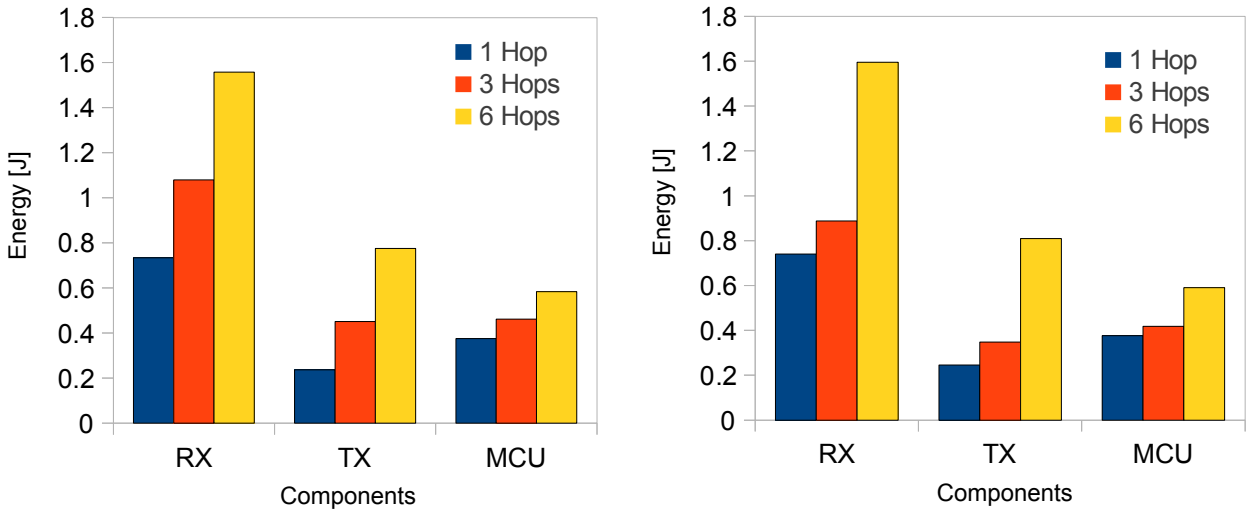
(c) Radio transmitting energy consumption with different data sizes

Figure 31: The overall energy consumption of the microcontroller, the radio receiving and the radio transmitting of all the network, composed of 6 sensor nodes, when using IPsec AH and ESP and without IPsec. ESP consumes more energy than AH and the energy consumption without IPsec is lower than with IPsec.

Energy Consumption using different number of hops

The overall energy consumption of the entire network shown in Figure 28, when using different number of hops, for *IPsec AH* and *ESP was* measured, for the *MCU*, the *TX* and the *RX*. Each experiment is executed utilizing *100 IP* datagrams of *16* bytes over *5* minutes.

Figure 32 *a* and *b* illustrate the energy consumption of the *MCU*, *TX* and *RX*, for *IPsec AH* and *ESP* accordingly, for *1*, *3* and *6* hops. As it can be observed, the energy consumption increase is relative proportional to the number of hops used, for both *AH* and *ESP*. Lower energy is consumed using one hop and more when *6* hops are used.



(a) Energy consumption for IPsec AH with different number of hops

(b) Energy consumption for IPsec ESP with different number of hops

Figure 32: The overall energy consumption of the microcontroller, the radio receiving and the radio transmitting of all the network, composed of 6 sensor nodes, when using 1, 3 and 6 hops for IPsec AH and ESP. Energy consumption is larger when 6 hops are used.

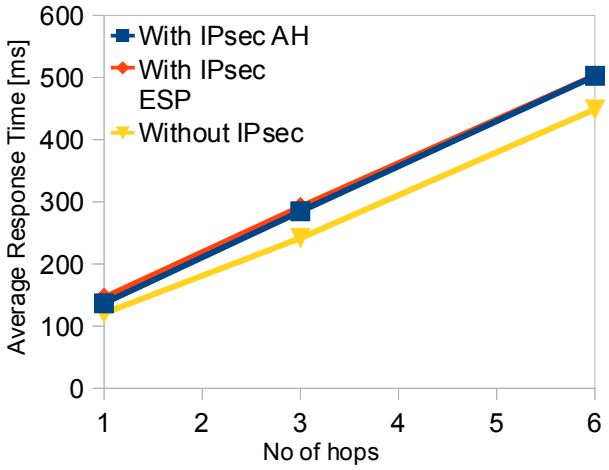
6.3.4 Delay

System-wide Response Time

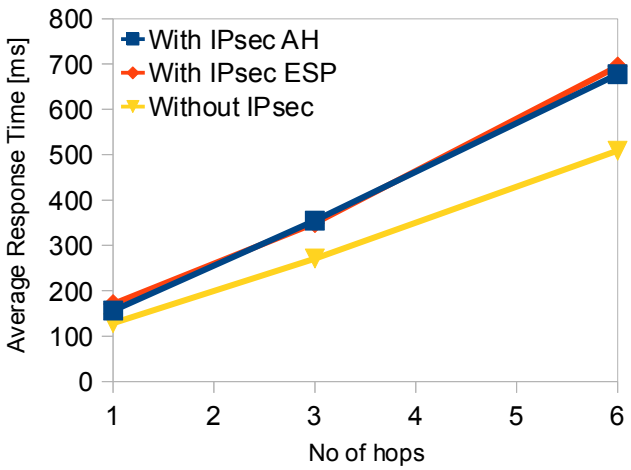
The response time for different data sizes with *IPsec AH* and *ESP* and without *IPsec* was measured. The experiments was executed using *1, 3* and *6* hops in the network illustrated in Figure 28.

The average response time in dependency of hop distance is shown in Figure 33. We observed that for a given data size, the overhead of both *AH* and *ESP* is constant, regardless of the number of hops. This is because the cost of forwarding the data with and without *IPsec* is the same for the intermediate sensor nodes. The overhead that is observed, is due to computation performed on the end nodes. In the worst case, i.e when using *IPsec ESP, 6* hops and a datagram size of *64* bytes, the overhead was *186 ms*.

The average response time in dependency of the *IP* datagram size is illustrated in Figure 34. The results show that *ESP* is slower than *AH*. This is because *AH* ensures authentication only, in contrast to *ESP* which authenticates plus encrypts and decrypts the messages. Moreover, when *IPsec* is not used, is faster than both *AH* and *ESP*. That was expected since less computations are performed.

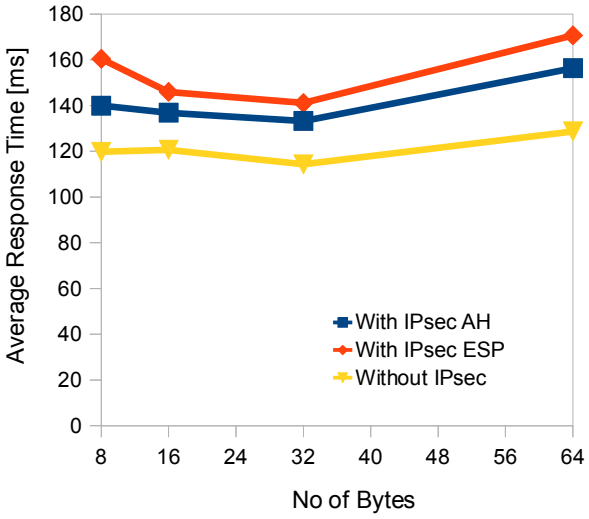


(a) Multi Hop with 16 bytes data size

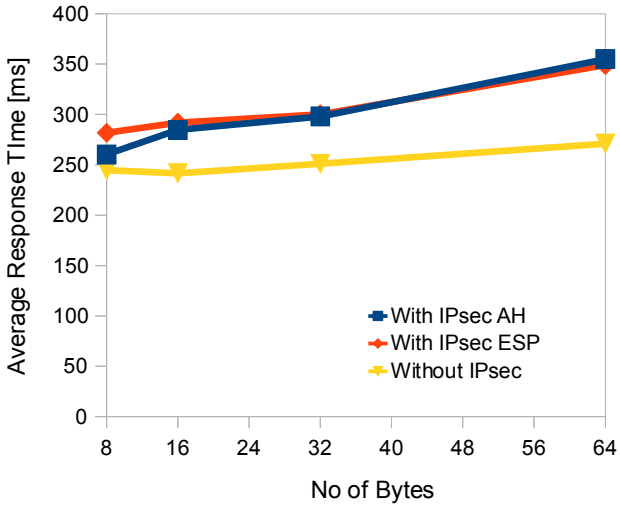


(b) Multi Hop with 64 bytes data size

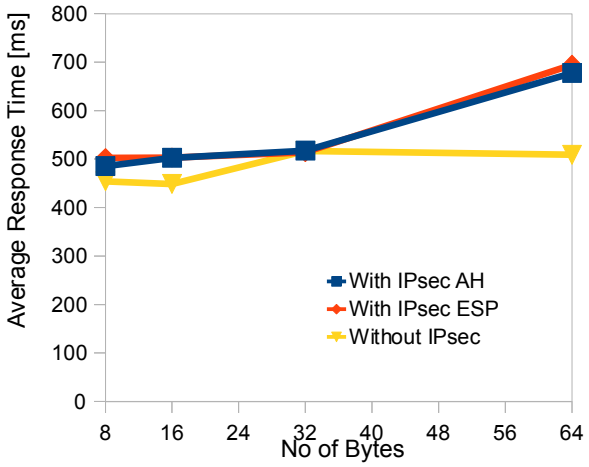
Figure 33: Average Response Time versus number of hops using IPsec AH and ESP and without IPsec. For both data sizes the overhead of IPsec is constant regardless the number of hops.



(a) Single Hop with different data sizes



(b) Multi Hop (3) with different data sizes



(b) Multi Hop (6) with different data sizes

Figure 34: Average Response Time versus datagram size using IPsec AH and ESP and without IPsec. AH is faster than ESP because it only authenticates the data, while ESP authenticates, encrypts and decrypts them.

Cooja Response Time

The response time for different number of hops, with *IPsec AH* and *ESP* and without *IPsec* was measured. The experiments were executed using *100 IP* datagrams of *16* bytes and the real time, inside *Cooja* simulator, needed for a message to be transferred form the *6LoWPAN* soft bridge to the analogous sensor node, was measured.

Figure 35 shows the average response time in dependency of the number of hops using a 16 bytes datagram size. As seen from the results, the overhead of both *IPsec AH* and *IPsec ESP* is constant across a single hop and a multihop network. Moreover, the difference of the average response time with and without *IPsec* is very small. Therefore, the overhead of *IPsec* is considered negligible.

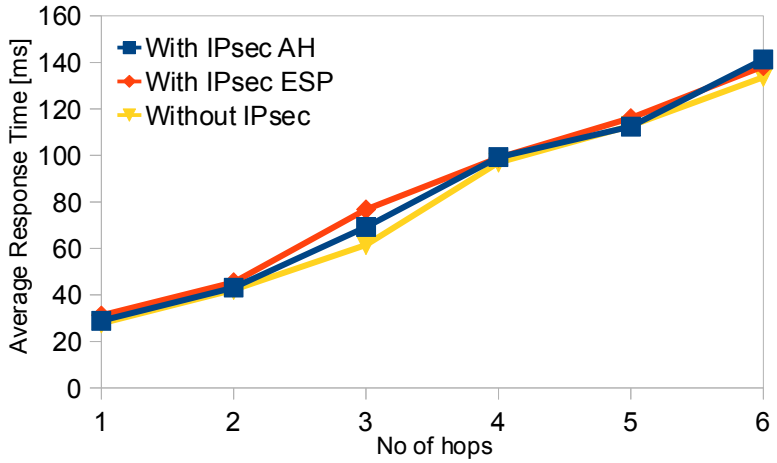


Figure 35: Average Response Time, inside Cooja simulator, versus number of hops using IPsec AH and ESP and without IPsec, and 16 bytes data size. The overhead of IPsec AH and ESP is negligible and constant despite the number of hops.

6.3.5 Packet loss

Tables 5, 6, and 7 show the packet loss for *100* datagrams using *1, 3, 6* hops accordingly, for *IPsec AH*, *IPsec ESP* and without *IPsec*. As it can be seen, when *AH* is used with *3* and *6* hops, there is packet loss with all the data sizes. When a single hop is utilized, there is no packet loss when the data size is *8* bytes. In contrast when *ESP* is used with *1* and *3* hops, there is packet loss only for a packet of *32* bytes and when *6* hops are utilized there is packet loss for packet of *16* and *32* bytes. Moreover, the biggest packet loss is *8%* and is observed when *ESP* is used with *1*, *3* or *6* hops for *32* bytes. In addition, without *IPsec* there is no packet loss using any number of hops.

	PACKET LOSS using 1 Hop		
No of bytes	*With IPsec AH*	*With IPsec ESP*	*Without IPSec*
8	0	0	0
16	7	0	0
32	3	8	0
64	5	0	0

Table 5: Packet loss for 100 datagrams using a single hop network and different data sizes for IPsec AH, IPsec ESP and without IPsec. Without IPsec, there is no packet loss and when AH is used there is packet loss at almost all the data sizes.

	PACKET LOSS using 3 Hop		
No of bytes	*With IPsec AH*	*With IPsec ESP*	*Without IPSec*
8	5	0	0
16	1	0	0
32	1	8	0
64	1	0	0

Table 6: Packet loss for 100 datagrams using 3 hops and different data sizes for IPsec AH, IPsec ESP and without IPsec. Without IPsec, there is no packet loss and when AH is used, for all the data sizes there is packet loss.

	PACKET LOSS using 6 Hop		
No of bytes	*With IPsec AH*	*With IPsec ESP*	*Without IPSec*
8	8	0	0
16	3	1	0
32	3	8	0
64	5	0	0

Table 7: Packet loss for 100 datagrams using 3 hops and different data sizes for IPsec AH, IPsec ESP and without IPsec. When AH is used, there is packet loss for all the data sizes, while without IPsec, there is no packet loss.

6.4 Network 2

6.4.1 Experimental Setup

The second network as it was outlined above, is composed of one Linux machine, running Ubuntu *11.10*, *13* Tmote sky sensor nodes and one *6LoWPAN* soft bridge that is implemented by another one Tmote sky sensor node and is illustrated in Figure 36.

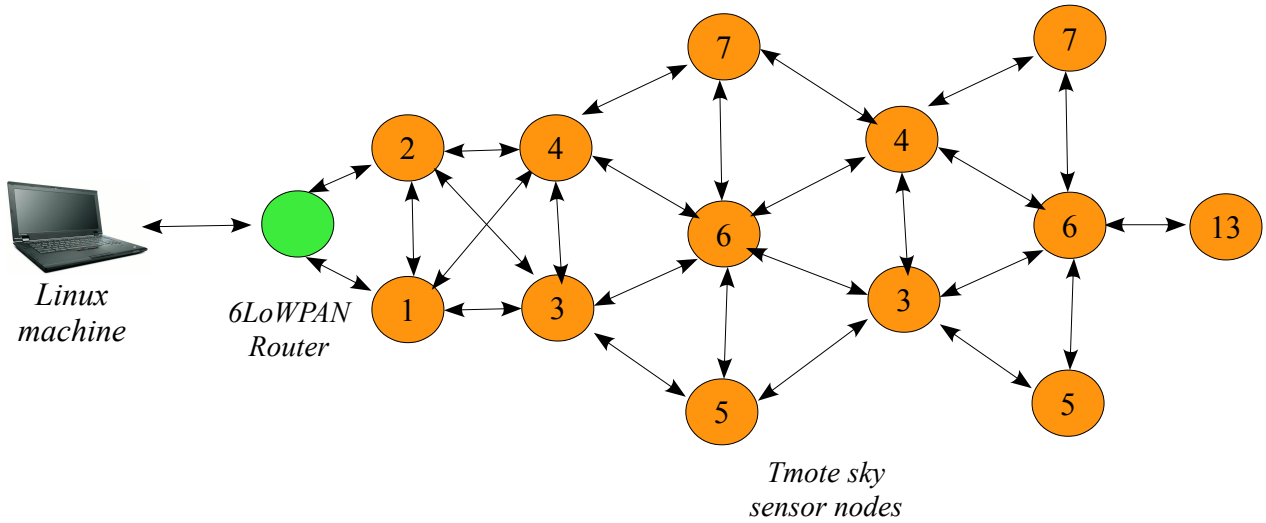
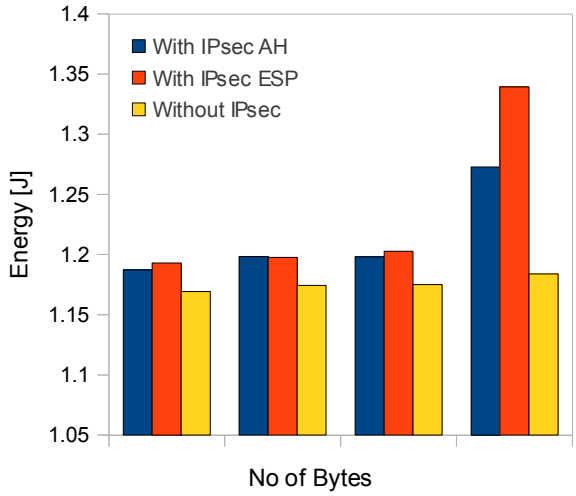


Figure 36: Network 2, composed of one Linux machine running Ubuntu 11.10, one 6LoWPAN soft bridge (implemented by one Tmote sky) and 13 Tmote sky sensor nodes.

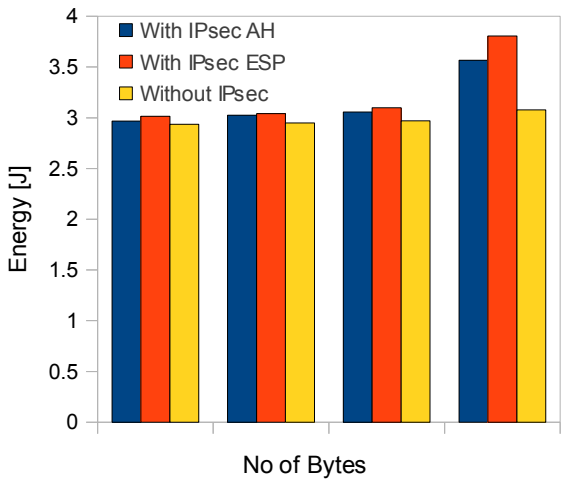
6.4.2 System-wide Energy Consumption

The overall energy consumption of all the network illustrated in Figure 36 when using *IPsec AH*, *IPsec ESP* and without *IPsec* was measured, for the *MCU*, the *TX* and the *RX*. Each experiment was run for *5* minutes and the messages were sent to the *13th* sensor node.

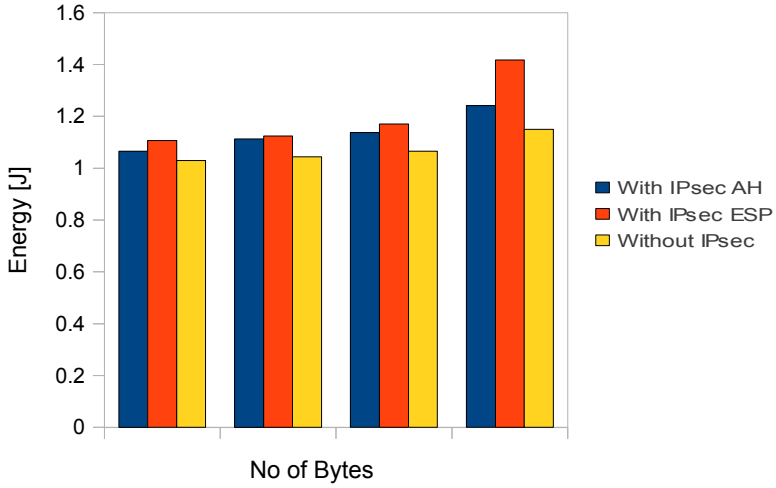
Figure 37 *a*, *b* and *c* show the energy consumption of the *MCU*, the *RX* and the *TX* accordingly. The results show that *ESP* consumes more energy than *AH* and this is because *AH* uses only authentication compared to *ESP* that uses both authentication and encryption. Moreover, energy consumption without *IPsec* is lower than with *IPsec*. In addition, the energy consumption is increased, as the datagram size grows. In the worst measured case, *IPsec ESP* on 64 bytes, the energy consumption is around *0.75*, *0.23* and *0.98 Joule* for *MCU*, *RX* and *TX* respectively.



(a) Microcontroller energy consumption with different data sizes



(b) Radio receiving energy consumption with different data sizes



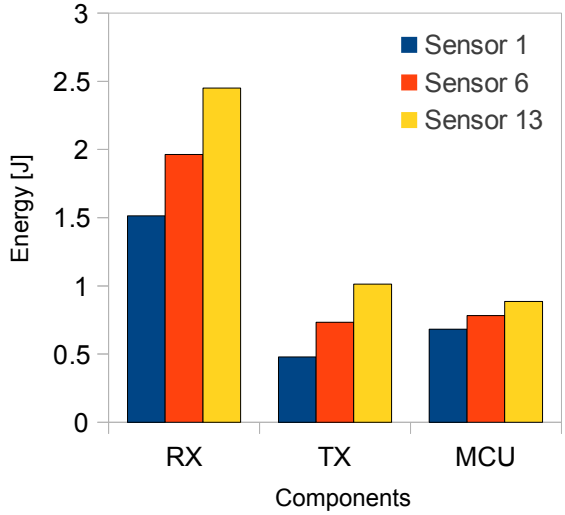
(c) Radio transmitting energy consumption with different data sizes

Figure 37: The overall energy consumption of the microcontroller, the radio receiving and the radio transmitting of all the network, composed of 13 sensor nodes, when using IPsec AH and ESP and without IPsec. ESP consumes more energy than AH and the energy consumption without IPsec is lower than with IPsec.

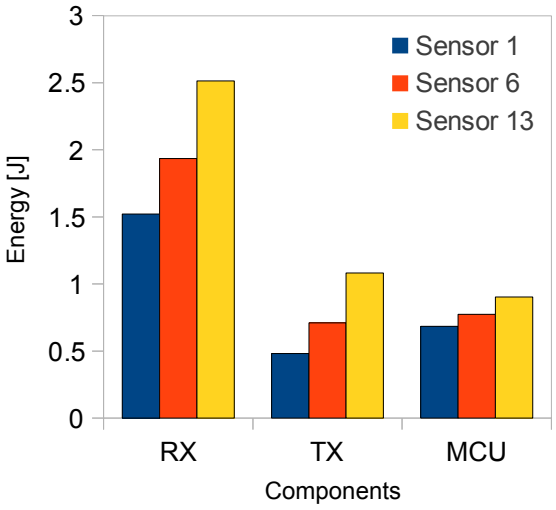
Energy Consumption using different number of hops

The overall energy consumption of all the network shown in Figure 36, using different number of hops, for *IPsec AH* and *ESP* was measured, for the *MCU*, the *TX* and the *RX*. Each experiment is executed utilizing *100 IP* datagrams of *16* bytes over *5* minutes.

Figure 38 *a* and *b* illustrate the energy consumption of the *MCU*, *TX* and *RX*, for *IPsec AH* and *ESP* accordingly, when sending the messages to sensor nodes *1*, *6* and *13*. As it can be observed, the energy consumption increase is relative proportional to the number of hops used, for both *AH* and *ESP*. The less energy is consumed using one hop and the most is consumed when sensor *13* was used.



(a) Energy consumption for IPsec AH with different number of hops



(b) Energy consumption for IPsec ESP with different number of hops

Figure 38: The overall energy consumption of the microcontroller, the radio receiving and the radio transmitting of all the network, composed of 13 sensor nodes, when sending a message to sensor node 1, 6 and 13 for IPsec AH and ESP. Energy consumption is higher when the message is sent to node 13.

6.4.3 Delay

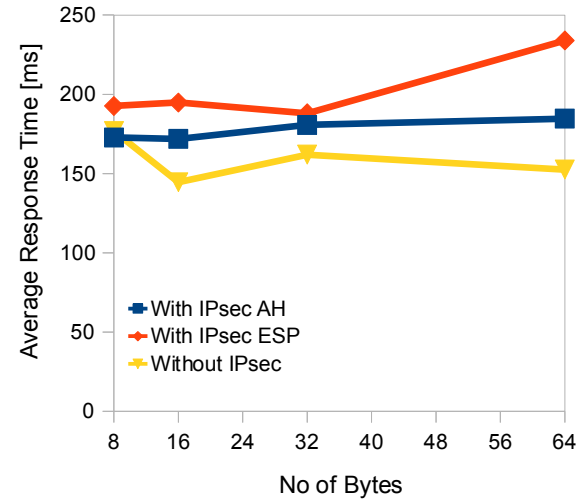
System-wide Response Time

The response time for different data sizes with *IPsec AH* and *ESP* and without *IPsec* was measured. The experiments were executed by sending messages to sensor nodes *1*, *6* and *13* in the network illustrated in Figure 36.

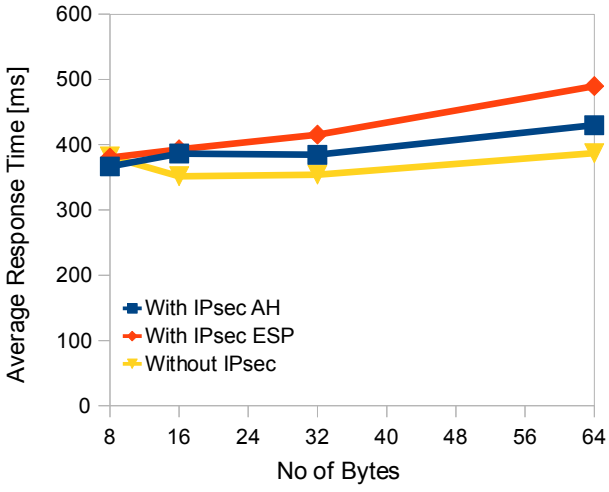
The average response time in dependency of the *IP* datagram size is shown in Figure 39. The results show that *AH* is faster then *ESP*. This is because *AH* ensures authentication only, compared to *ESP* which authenticates, encrypts and decrypts the messages. In addition, when *IPsec* is not used, is faster than both *AH* and *ESP,* since less computations are performed without *IPsec*.

Figure 40 illustrates the average response time in dependency of hop distance. It can be observed that for a given data size, the overhead of both *AH* and *ESP* is constant, regardless the number of hops. This is because the cost of forwarding the data with and without *IPsec* is the same for the intermediate sensor nodes. The overhead that was observed, is due to computation performed on the end nodes. In the worst case, i.e when

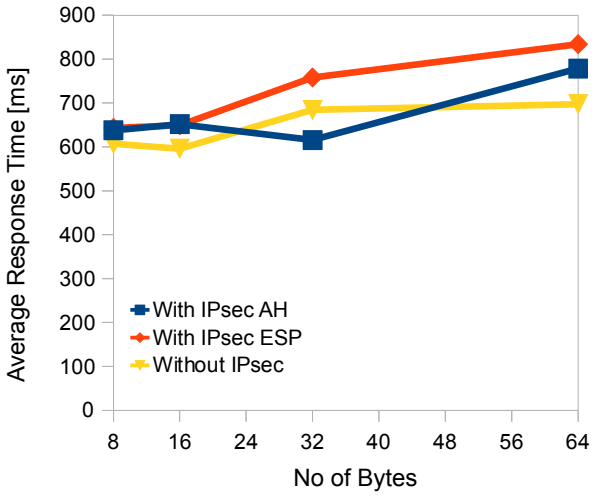
using *IPsec ESP* and sending a datagram of size *64* bytes to the sensor node *13*, the overhead is *137 ms*.



(a) Single Hop with different data sizes

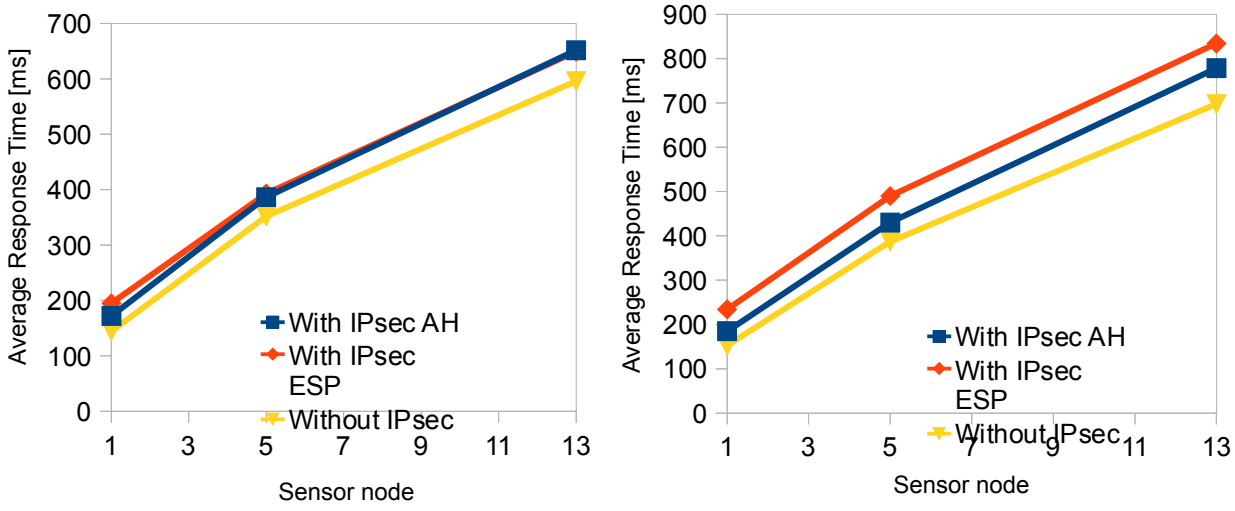


(b) Multi Hop (sensor 6) with different data sizes



(c) Multi Hop (sensor 13) with different data sizes

Figure 39: Average Response Time versus datagram size using IPsec AH and ESP and without IPsec. AH is faster than ESP because it only authenticates the data, while ESP authenticates, encrypts and decrypts them.



(a) Multi Hop with 16 bytes data size

(b) Multi Hop with 64 bytes data size

Figure 40 : Average Response Time versus number of hops using IPsec AH and ESP and without IPsec. For both data sizes the overhead of IPsec is constant regardless the number of hops.

Cooja Response Time

The response time for different number of hops, with *IPsec AH* and *ESP* and without *IPsec* was measured. The experiments were executed using *100 IP* datagrams of *16* bytes, and the real time inside *Cooja* simulator, needed for a message to be transferred form the *6LoWPAN* soft bridge to the analogous sensor node, was measured.

Figure 41 illustrates the average response time in dependency of the number of hops using a *16* bytes datagram size. As it can be observed from the results, the overhead of both *IPsec AH* and *IPsec ESP* is constant, regardless the number of hops of the network. Moreover, the difference of the average response time with and without *IPsec* is very small and thus, the overhead of *IPsec* is considered negligible.

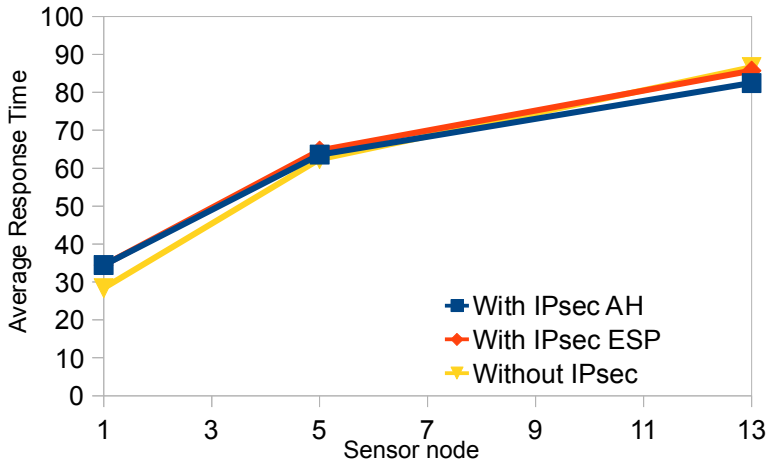


Figure 41 : Average Response Time, inside Cooja simulator, versus number of hops (sensor 1, 6 and 13) using IPsec AH and ESP and without IPsec, and 16 bytes data size. The overhead of IPsec AH and ESP is negligible and constant despite the number of hops.

6.4.4 Packet loss

Tables 8, 9, and 10 show the packet loss for *100* datagrams when sending to *1, 6* and *13* sensor node accordingly, for *IPsec AH*, *IPsec ESP* and without *IPsec*. As it can be seen, when *AH* is used there is packet loss with all the data sizes for the three sensor nodes. In contrast when *ESP* packet loss is observed only for *32* and *64* bytes data size. Moreover, the biggest packet loss is *8%* and is observed when *ESP* is used with the three sensor nodes, for *32* bytes data size. In addition, without *IPsec* there is only one packet loss with the sensor node *13* for *32* bytes data size.

	PACKET LOSS (sending to sensor 1)		
No of bytes	*With IPsec AH*	*With IPsec ESP*	*Without IPSec*
8	4	0	0
16	3	0	0
32	3	8	0
64	7	2	0

Table 8: Packet loss for 100 datagrams using a single hop network and different data sizes for IPsec AH, IPsec ESP and without IPsec. Without IPsec, there is no packet loss and when AH is used there is packet loss for all the data sizes.

	PACKET LOSS (sending to sensor 6)		
No of bytes	*With IPsec AH*	*With IPsec ESP*	*Without IPSec*
8	2	0	0
16	2	0	0
32	5	8	0
64	1	4	0

Table 9: Packet loss for 100 datagrams and different data sizes, when sending to sensor node 6, for IPsec AH, IPsec ESP and without IPsec. Without IPsec, there is no packet loss and when AH is used, for all the data sizes there is packet loss.

	PACKET LOSS (sending to sensor 13)		
No of bytes	*With IPsec AH*	*With IPsec ESP*	*Without IPSec*
8	5	0	0
16	4	0	0
32	5	8	1
64	3	0	0

Table 10: Packet loss for 100 datagrams and different data sizes, when sending to sensor node 13, for IPsec AH, IPsec ESP and without IPsec. When AH is used, there is packet loss for all the data sizes, while ESP has 8% and without IPsec has 1% packet loss for *32* bytes data size.

6.5 Memory footprints

The memory footprints of the extended *IPsec* implementation presented here using *MIRACL AES*, *TI AES* and a hardware *AES* implementation were measured and they are illustrated in Table 11. In addition *AES-XCBC-MAC-96* mode of operation was utilized for *AH,* and *AES-CTR* and *AES-XCBC-MAC-96* for encryption and authentication in *ESP*, accordingly. The *RAM* footprint overhead ranges from only *0.1* to *0.4 KB*. The lower *RAM* overhead, that is *0.1 KB*, is observed when hardware *AES* is used, and when *MIRACL AES* is utilized the bigger overhead was seen, that is *0.4 KB* for both *AH* and *ESP*. Furthermore, the *RAM* overhead is *0.3 KB* when *TI AES* is used for both *AH* and *ESP*. Hence system *RAM* usage is under the Tmote sky *RAM* usage, i.e *10 KB*. The *ROM* usage lies between *2.3 KB*, when *AH* with hardware *AES* is used, to *5.7 KB*, when *ESP* with *TI AES* is utilized. Therefore, system footprint is kept always under *48 KB*, which is the Flash *ROM* size of the Tmote sky sensor. Consequently, both *IPsec AH* and *ESP* can be embedded in resource-constrained embedded devices and allow space for applications as well.

	ROM (KB)		*RAM (KB)*			
System	*overall*	*diff*	*data*	*bss*	*overall*	*diff*
Without IPsec	37.9	-	0.2	6.8	7.0	-
AH with MIRACL AES	42.9	5	0.2	7.2	7.4	0.4
AH with TI AES	43.4	5.5	0.2	7.1	7.3	0.3
AH with Hardware AES	40.2	2.3	0.2	6.9	7.1	0.1
ESP with MIRACL AES	43.1	5.2	0.2	7.2	7.4	0.4
ESP with TI AES	43.6	5.7	0.2	7.1	7.3	0.3
ESP with Hardware AES	40.4	2.5	0.2	6.9	7.1	0.1

Table 11: Memory footprints illustrate that ROM and RAM overhead for both IPsec AH and ESP is small and therefore, they can be embedded in constrained devices.

6.6 Critical evaluation

As outlined above, two networks were used for the evaluation of the *IPsec,* one with *6* and one with *13* Tmote sky sensor nodes. These two networks were selected for two main reasons. Firstly, to compare the performance of *IPsec AH*, *IPsec ESP* and without *IPsec*, and secondly to examine how the number of sensor nodes exist in the network, as well as the density of the network, affect the results.

Firstly the network composed of *6* sensor nodes was used. The first experiment was executed in order to evaluate the performance of two different *AES* implementations, namely, *MIRACL AES* and *TI AES,* in terms of system-wide response time and packet loss. The results showed that *MIRACL* is marginally faster than *TI,* and that the packet loss is unrelated to *AES* implementation. However, the results regarding the response time is not as accurate due to the reasons explained above. In addition, the response time of each packet is affected from the *CSMA MAC* driver, that retransmits packets after a collision detect. Moreover, packet loss is affiliated with the *ContikiMac* driver, which keeps the radio off as much as possible and periodically checks for radio activity in the radio medium.

Afterwards, many experiments were performed to examine the energy consumption of *IPsec,* in terms of *MCU*, *TX* and *RX*. It can be observed that *IPsec AH* consumes less energy than *ESP* for all the data sizes. This is due to the fact, that *AH* only authenticates the data, whereas *ESP* uses both authentication and encryption, thus more computations are performed. In addition, energy consumption increase is relative proportional to the number of hops used, for both *AH* and *ESP*, since as the number of hops increase, more

sensor nodes are involved in the transmission of the message to the appropriate sensor node. If a node is participating in the message transmission, it requires more computations and thus it consumes more energy than a node that does not participate. Therefore, the overall system energy consumption is increased. Furthermore, *IPsec* consumes more energy than without *IPsec* and this was expected, since without *IPsec,* authentication, encryption and decryption are not executed.

Then, the average response time was measured in two ways as explained earlier, due to the fact that one is not accurate, because the calculated time belongs to two different spacetimes. However, it is useful since it shows the relative delay of the network. The other, is utilized in order to estimate the real time needed inside *Cooja* simulator. The results regarding the system-wide response time showed that *IPsec AH* is faster than *ESP*, because *AH* ensures authentication only, whereas *ESP* authenticates, encrypts and decrypts the messages, and hence more computations take place. Moreover, in both system-wide and *Cooja* response time, the results showed that the overhead of both *IPsec AH* and *ESP* is constant across a single hop and a multihop network. In addition, the overhead observed in *Cooja* response time seems to be negligible, since is too small for both *AH* and *ESP*. Moreover, one factor that affects the response time is the *CSMA MAC* driver, that retransmits packets after the detection of a collision .

Subsequently, the packet loss for *100* packets was measured. The results illustrate that *IPsec AH* has packet loss whatever the size of the packet is, while *IPsec ESP* is more likely to has packet loss when the data size is *32* bytes. On the other hand, without *IPsec*, packet loss is very rare. In addition, as outlined above, *ContikiMac* driver affects the number of packet loss.

The same experiments have been executed using the second network in order to study if the results are influenced by the number of sensor nodes that exist in the network, and the density of the network. The results showed that the performance of *IPsec* is neither affected from the number of sensors the network is composed of, nor from the density of the network. This was deduced by the fact that the result extracted for this network are similar to the results found on the first network. That is, *IPsec AH* is slightly more efficient than *IPsec ESP*. Nevertheless, the system-wide energy consumption of the first network is less than the system-wide energy consumption of the second network. That was expected, since the second network is composed of more sensor nodes than the first network. In addition, the average response time needed for a message to be sent to a sensor node in the first network is less than the time needed for a message to be sent to a sensor node in the second network, using the same number of hops. For instance, the average time needed for a message of *64* bytes data size to be sent to a sensor using *6* hops, i.e sensor *6* of the first network and sensor *13* of the second network (Figures 28 and 36 respectively), is around *700 ms* for the first network and around *800 ms* for the second network, when *ESP* is utilized. This happens, due to the fact that the intermediate nodes in the second network, communicate not only with their neighbor which is in the path for the transmission of the message, but with all the other neighbors they have, exchanging other messages like messages regarding the routing. A sensor node, can only transmit one packet at each time and thus, message transmission is delayed more. Therefore, *IPsec* performs in the same way no matter of the kind of the network (simple or more complex network), in terms that *AH* is slenderly more efficient than *ESP*, but the overhead is relative proportional to the number of sensor nodes used and the density of the network.

Finally, memory footprints of the *IPsec* implementation were measured. As it can be observed from the results, the *RAM* overhead is very small and the *ROM* usage is always under *48 KB*, which is the *Flash ROM* size of the Tmote sky sensor. Therefore, both *IPsec AH* and *ESP* can be integrated in constrained devices and allow space for applications as well.

In conclusion, the results show that it is feasible and efficient to use *IPsec* to secure communication between hosts in the Internet and sensor nodes.

7. DTLS

7.1 TLS/DTLS

7.1.2 Introduction

In order to investigate the *DTLS* implementation found during my research, a detailed study of Transport Layer Security (*TLS*) protocol [30] and Datagram Transport Layer Security (*DTLS*) [31] protocol needed to be done. *TLS* is based on the Secure Sockets Layer version 3 (*SSL3.0*) protocol and operates between transport-layer and application layer. Therefore, it provides privacy and data integrity between two communicating applications, but it can only be used over *TCP*. More specifically this protocol is constructed in such a way in order to avoid tampering, eavesdropping and message forgery during the communication of two client/server applications. Furthermore, *TLS* can be used in two modes, mutual authentication and server-side authentication. In mutual authentication, both server and client are authenticated, in contrast to server-side authentication where only server needs to be authenticated. Moreover, the protocol consists of four (sub-) protocols, namely the Record protocol, the Alert protocol, the Handshake protocol and the Change Cipher Spec. protocol.

7.1.3 Record protocol [30]

Record protocol provides private and reliable connection, using symmetric cryptography for data encryption and secure hash functions for *MAC* computations, that is utilized for the computation of the message integrity check. All the messages between the client and the server are transferred via this protocol. A given record can be modelled as: $R=M \,||\, A \,||\, P$ where, M is the message (e.g the payload or record content), A is an optional message authentication field, P is an optional message padding field and || denotes concatenation.

7.1.4 Alert protocol [30]

Alert messages transfer the strictness of the message (fatal or warning) and a description of the alert, and can be sent at any time during the handshake. When an endpoint detects an error, it sends an alert message to the other endpoint in order to notify it about this event. These messages are encrypted and compressed like the other messages.

7.1.5 Handshake protocol [30]

The Handshake protocol is used to perform authentication between the client and the server and to allow them to negotiate an encryption algorithm and cryptographic keys. Hence, three basic security properties are accomplished:

- Authentication of the peer's identity using asymmetric (public-key) cryptography.
- Secure negotiation of a shared secret: The secret cannot be obtained even by a man-in-the-middle attack for any authenticated connection and is unavailable to eavesdroppers as well.
- Reliable negotiation: If an attacker alters the negotiation communication he would be detected by the parties (client/server) of the communication.

Steps of the Handshake Protocol (shown in Figure 42):

- Exchange hello messages in order to agree on algorithms, exchange random values and session resumption check.
- Exchange cryptographic parameters so that client and server are able to agree on a premaster secret.
- Exchange certificates and cryptographic information so that client and server are able to authenticate themselves.
- Generation of a master secret from the premaster secret and exchanged random values.

- Procure security parameters to the record layer.
- Verification from the client and the server that the security parameters their peers have calculated are the same and that the handshake has not being tampered by an attacker .

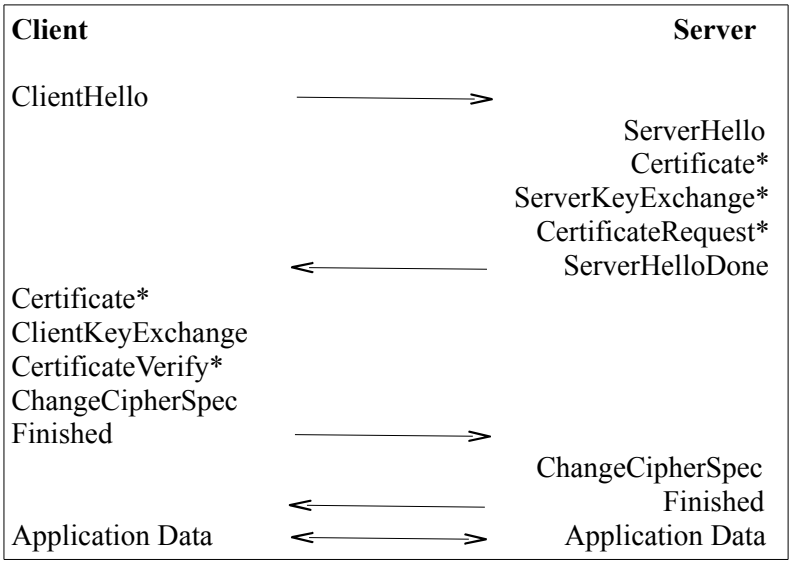


Figure 42: Message flow for a full handshake (* indicates optional messages that are not always sent) (From [30].)

7.1.6 Change Cipher Spec. protocol [30]

Both the client and server send the change cipher spec message to notify each other that the following records will be protected under the *CipherSpec* and keys that have being just negotiated. Moreover, it updates the cipher suite to be used in the connection and allows a change in the *TLS* session without the renegotiation of the connection.

Cipher suite is an agreed configuration composed of :

1. A protocol version e.g TLSv1.0, TLSv1.2
2. A key exchange algorithm e.g ECDH, RSA, DHE
3. An authentication algorithm e.g ECDSA, RSA
4. A bulk encryption algorithm e.g AES, DES
5. A message authentication algorithm e.g SHA1, SHA2 MD5

For instance the string TLSv1.2_ECDH_RSA_WITH_AES_128_CBC_SHA1 describes a cipher suite given below according to the color coding:

1. a TLS version 1.2 of the protocol is used,
2. elliptic curve Diffie-Hellman will be used for key exchange,
3. RSA will be used for authentication,
4. AES (with a 128-bit key in CBC mode) will be used for bulk encryption, and
5. a SHA1-based MAC will be used to ensure message authenticity.

DTLS protocol is based on the *TLS* protocol and provides similar security properties. They have many similarities and use the same cipher suites. The difference between the two protocols is that *TLS* can only be used over *TCP*, in contrast to *DTLS* which can be used over *UDP*.

7.2 Analysis of the Contiki DTLS implementation

As stated above, the already existing implementation of *DTLS* for the *Contiki OS,* outlined in chapter 3 in section 3.1.2, was selected to be examined, and to check if it can be loaded on the Tmote sky platform which is an ultra low power wireless sensor board.

This implementation supports the DTLS_PSK_WITH_AES_128_CCM_8 cipher suite, which is one of the mandatory suites to develop, regarding to the Constrained Application Protocol (*CoAP*) Internet Draft [45]. In order to have as less memory requirements as possible, it does not support any of the optional messages, that is certificate-related messages.

DTLS for *Contiki OS* is available as a *Contiki* internal library and hence can be easily used in any application by including *dtls.h*. *DTLS API* was developed as self-explanatory and straightforward as possible. Some of the functions it produces are read and write functions for both server and client, as well as *DTLS Listen* and *DTLS Connect* for server and client accordingly.

Moreover, this implementation was integrated into the Network Configuration Protocol Light (*NETCONF-Light)* implementation [58], and this integration follows the optional support of *DTLS Pre-Shared Key* (*PSK*) authentication and utilizes *DTLS PSK* cipher suites outlined in [59].

This implementation works as follows: When a client connects to the server, *DTLS* handshake should be performed, and therefore client waits until the completion of it. When the handshake is finished, client is notified through a special event, which is provoked after handshake completion. At this time, client has in its possession all the necessary security information, that will be used while sending and receiving data.

During the *DTLS* handshake, a "*PSK identity*" is included in the *DTLS ClientKeyExchange* message by the client, as a means to declare which key will be used. This "*PSK identity*" is used on the server side to search for the key that corresponds to the provided "*PSK identity*". Afterwards, if the two *PSKs* match, the client is authenticated, and the *NETCONF* username, which is utilized as the authenticated identity of a *NETCONF* client, is derived using the "*PSK identity*".

Furthermore, the application which includes *DTLS* can operate either as *DTLS* server or *DTLS* client, but not as both simultaneously. *PSK*, which exists on the mote, is located in the Flash memory and is accessed via *Contiki's File System API*. *SHA256* and *HMAC_SHA256* are calculated using an already existing external library described in [26], and libraries of the *OpenSSL* project [60] with slight modification are utilized, for *AES* operations. In addition, all static variables that *AES* needs are stored in Flash memory, to reduce the amount of the *RAM* usage.

As outlined above, this implementation is investigated in terms of memory footprint. Table 12 illustrates the amount of *RAM* and *ROM* required for both *DTLS* server and *DTLS* client. Server needs *12492* bytes *RAM* and *64210* bytes *ROM* and client *12532* bytes *RAM* and *65232* bytes *ROM*. Taking into consideration that Tmote sky is composed of *10 KB RAM* and *48 KB ROM*, i.e *10240* bytes *RAM* and *49152* bytes *ROM*, it can be observed that *DTLS* server has *2252* bytes *RAM* and *15060* bytes *ROM* overheads, and *DTLS* client has *2292* bytes *RAM* and *16080* bytes *ROM* overheads. Consequently, this implementation cannot be loaded in the Tmote sky board as it is.

System	*RAM (bytes)*	*ROM (bytes)*
DTLS server	12492	64212
DTLS client	12532	65232

Table 12 Memory footprint for DTLS

7.3 Implementation

Since the implementation cannot be loaded in the Tmote sky platform, for the reasons presented above, optimizations to reduce both *RAM* and *ROM* requirements were necessary. Firstly, the *Contiki OS'* firmware size was reduced.

This was achieved by reducing the *uIP/IPv6* stack size and by changing the Radio Duty Cycling (*RDC*) protocol. *TCP* and *UDP* can be individually enabled or disabled. *TCP* is not needed since *DTLS* works only over *UDP*. Therefore, *uIP/IPv6* stack size was reduced by disabling the *TCP* which eliminates a *MSS-sized* buffer (i.e a maximum segment size buffer). Additional *RAM* is required by the *RDC* protocol in order to store neighbor information. Hence, the sequence number array size that is utilized for duplicate packet detection was reduced. Neighbor cache size and routing table size were decreased from *12* to *4*. Furthermore, the *RDC* phase optimization was turned off, in order to eliminate the neighbor wake time table. During the phase optimization a mechanism is used in the transmitter in order to keep track the wake periods of the receiver. Thus, by disabling the phase optimization *RAM* usage is decreased, since the wake periods of the receiver are not stored anymore.

One other essential optimization was the reduction of the *queuebuf* size. Queue buffers are stored in *RAM*. Hence, by reducing the maximum number of buffers that can be stored, the *RAM* usage is decreased as well. *Queuebuf* was decreased from *12* to *4*.

Moreover, the *NullRDC* driver was used instead of the *ContikiMAC* driver, since using *NullRDC* driver all nodes keep their radio turned on all the time. Thus, there is no need to store extra information in order to be able to know when to switch on/off their radio.

In addition, *NullMAC* driver was used, instead of the *CSMA* driver. *NullMAC* does not retransmit packets after a collision is detected, while *CSMA* does. Hence, no additional space is required to store information about the packets that need retransmission.

Furthermore the buffer size was decreased from *512* to *240* bits and the *UDP* connections from *12* to *4*. By altering the *Contiki OS'* firmware size the *RAM* overhead was eliminated, but *ROM* overhead still remained.

As a means to eliminate the *ROM* overhead, the *AES* implementation was removed and a dummy *AES* implementation was used instead. This is an acceptable practice, due to the fact that when the implementation will be used on a real sensor, a hardware *AES* implementation will be used as well. By removing the *AES* implementation *ROM* usage was decreased by *11814* bytes. However, there was still *ROM* overhead. Therefore, the *RPL* protocol was removed, in order to overcome the overhead. By removing the *RPL* protocol the *ROM* usage was reduced by *10328* bytes. After these optimizations the *ROM* overhead was successfully eliminated.

The *RAM* and *ROM* requirement of the optimized *DTLS* implementation were measured by running *Contiki* in the *MSPSim* emulator [60]. *RAM* is computed as the sum of the runtime stack usage and the global data. Furthermore they compared with the memory footprints of the already existing *DTLS* implementation.

The memory footprints of the default and the optimized *DTLS* implementation are shown in Table 13. Optimized *DTLS* server needs *9302* bytes *RAM* and *38562* bytes *ROM*, while *DTLS* client requires *9342* bytes *RAM* and *39508* bytes *ROM*. Therefore, optimized *DTLS* server needs *3190* bytes *RAM* and *25650* bytes *ROM,* less than the default *DTLS* server, and optimized *DTLS* client needs *3190* bytes *RAM* and *25724* bytes *ROM,* less than the default *DTLS* client. Hence, the optimized implementation can be loaded in a Tmote sky platform, but the amount of *RAM* memory that is remaining free is very small. More specifically *DTLS* server and *DTLS* client has *938* and *898* bytes free respectively.

System	RAM (bytes)	ROM (bytes)
DTLS server	12492	64212
DTLS client	12532	65232
Optimized DTLS server	9302	38562
Optimized DTLS client	9342	39508

Table 13: Memory footprint for DTLS

The above modifications were tested using *Cooja* simulator by running an experiment composed of one *DTLS* server and one *DTLS* client. However, during the handshake process an error with regards to stack overflow occurred. Due to time constraints of the project stack debugging was not performed. This process is very difficult and time consuming and would restrict the efficient execution of the evaluation.

7.4 Critical evaluation

As stated above, the already existing *DTLS* implementation could not be loaded in the Tmote sky platform due to its memory requirements. Therefore, many optimizations took place in order to reduce the *ROM* and *RAM* usage.

As outlined, when the optimized implementation was tested using *Cooja* an error with respect to stack overflow occurred during the handshake process. This implies, that even though the *RAM* overhead was eliminated, the amount of *RAM* memory that remains free on the Tmote sky is not enough for the communication of the two sensors.

At this time, it is worth mentioning that according to [58], NETCONF should be running fine in constrained devices which have about 50 KB of RAM and 250 KB of ROM (not so constrained devices) and it may be running in devices which have about 10 KB of RAM and 100 KB of ROM, with very little resources left for other application code. Considering that NETCONF is integrated with this implementation, very few resources are left and that's why stack overflow error occurred.

One additional drawback of this implementation is that during the optimization phase, the *RPL* protocol was removed, which has been established as the *de facto* standard for routing in *6LoWPAN*. Therefore, even though the provided *DTLS* implementation is presented as lightweight, it cannot be loaded to the Tmote sky platform not even after the described optimizations.

8. Comparison of IPsec and DTLS in 6LoWPAN

Table 14 shows the memory footprints of *IPsec*, the default *DTLS* and the optimized *DTLS* implementation. As it can be observed, default *DTLS* implementation has by far the most memory requirements and it cannot be loaded to the Tmote sky platform, as described above. More specifically, *DTLS* server requires *62.8 KB ROM* and *12.2 KB RAM* and *DTLS* client *63.8 KB ROM* and *12.2 KB RAM*. Nevertheless, both optimized *DTLS* server and client have slightly less *ROM* requirements than *IPsec* (for any *IPsec* protocol). However, the *RAM* usage is higher in both optimized *DTLS* server and client than *IPsec*. In contrast, as found above, *IPsec* can be efficiently implemented in constrained devices.

System	*ROM (KB)*	*RAM (KB)*
AH with MIRACL AES	42.9	7.4
AH with TI AES	43.4	7.3
AH with Hardware AES	40.2	7.1
ESP with MIRACL AES	43.1	7.4
ESP with TI AES	43.6	7.3
ESP with Hardware AES	40.4	7.1
DTLS server	62.8	12.2
DTLS client	63.8	12.2
Optimized DTLS server	37.7	9.1
Optimized DTLS client	38.6	9.1

Table 14: Memory footprints of IPsec, default DTLS and optimized DTLS

Alshamsi A. and Saito T. in [61] provided a technical comparison of *IPsec* and *SSL/TLS*. As stated in section 7.1 *DTLS* protocol is based on the *TLS* protocol and provides similar security properties. In addition, they utilize the same cipher suites and have many similarities. Thus, many technical characteristics of *TLS* and *DTLS* are similar and are described below.

IPsec supports only mutual authentication, where both endpoints must be authenticated, compared to *SSL* that supports mutual authentication, server-side authentication, where only the server should be authenticated, and anonymous where none server and client should be authenticated [61]. In the context of *6LoWPAN* it is better to authenticate both endpoints because sensor nodes should send their data to a trusted server or host on the Internet, and the receiver should only accept data from a trusted endpoint. This is due to the fact that *6LoWPAN* are used in applications that monitor physical or environmental conditions or in automation applications in home, office and factory environments. Therefore, the fact that *IPsec* does not support server-side or anonymous authentication does not pose a problem.

Moreover, *IPsec* can be used either in tunnel mode, where the connection is established between Gateway-to-Gateway, Gateway-to-Host and Host-to-Host, or in transport mode where the connection is established between Host-to-Host. However, tunnel mode is not practical in *6LoWPAN* because the additional headers would further increase the packet size. In *SSL* there is one connection per one session type, where each session is independent, but as the number of sessions increases, the throughput might fall down [61]. This is a disadvantage in the context of *6LoWPAN* due to the fact that more *MCU* cycles would be required in order to establish each connection.

Furthermore, the order of cryptographic operations in *IPsec* and *SSL* differs. *IPsec* first encrypts the data and then creates the *MAC* for the encrypted data, in contrast to *SSL* where the *MAC* is created first for the plaintext and then the data is encrypted. Therefore, *IPsec* would first verify the *MAC* and then perform the decryption, compared to *SSL* where the decryption would take place before the verification of the *MAC* and this could waste *CPU* when the *MAC* is not valid [61]. This is an essential drawback for *DTLS* in the context of *6LoWPAN*. The algorithms used in *6LoWPAN* should be as efficient and optimized as possible. It is not practical to waste *MCU* cycles in order to decrypt invalid data.

In addition, *DTLS* is implemented at the application level while *IPsec* is implemented at the kernel level and thus, the support in multiple environments is easier. Moreover, managing *IPsec* policy and deployments is complex. However, due to the fact that *IPsec* resides in network layer any transport or application on top of network layer is protected.

In conclusion, *IPsec* can be used over *TCP*, *UDP* and *ICMP* in network layer, providing connectionless integrity, data origin authentication and confidentiality. In contrast, *DTLS* can only be used over *UDP* and does not work on resource-constrained embedded devices such as Tmote sky due to its memory requirements, as explained above. Therefore, it is more beneficial to use *IPsec* instead of *DTLS* for secure communications in *6LoWPAN*.

9. Conclusions

9.1 Critical Evaluation

WSNs and traditional *IP* networks have already begun to be integrated using the *IPv6* protocol and *6LoWPAN*. Hence, *6LoWPAN* will be part of the evolution of the Internet. As sensing applications are connected to the Internet, the need for secure end-to-end communications between sensor nodes and Internet host is increasing. Therefore, appropriate security techniques must be adopted in the context of *6LoWPAN*.

As outlined in the beginning of this study, this project had two aims. The first aim was to conduct an in-depth evaluation of the performance of *IPsec* in *6LowPAN* and the second aim was to investigate if end-to-end security can be efficiently implemented using other appropriate methods.

In order to achieve the aims of the project a detailed background research was done. Firstly, traditional *WSNs* were described and afterwards, *6LoWPANs* were identified and distinguished from them. Subsequently, an introduction to *IPv6* took place, where the architecture, the new provided features compared to *IPv4* and the extension headers were presented. In addition, a thorough explanation of *IPsec* took place, where the two security protocols (*AH* and *ESP*) used in *IPsec* and the security associations were described with detail. Furthermore, *Contiki OS*, *Cooja* simulator and Tmote sky sensor node, which were used for the implementation and the evaluation of the investigated methods of this project, were outlined.

Moreover, a research about the various methods that have been utilized by other researchers throughout the years, providing end-to-end security on either the traditional *WSNs* or the *6LoWPANs* was performed. This was in order to examine if any other method, except *IPsec* can be efficiently used in *6LoWPANs*.

The project aims were fulfilled. Firstly, by investigating the already existing *IPsec* implementation and extending it in order to correctly work under the new specifications of the *6LoWPAN*. This extension was the most important factor for the achievement of this project, since it was critical for evaluating the performance of *IPsec* in *6LoWPAN*. In addition, a detailed study of *TLS/DTLS* protocols took place in order to be able to examine the *DTLS* implementation that already exists in *6LoWPAN* and was selected during the research. Afterwards, the already existing *DTLS* implementation was analyzed and optimized in order to load it in the Tmore sky platform, which is an ultra low power wireless sensor board and it is utilized for the evaluation of the performance of the techniques that are used in this project.

Moreover, an in-depth evaluation of the performance of the extended *IPsec* implementation was accomplished by running various experiments using two different networks. In addition, memory footprints of the existing and the optimized *DTLS* implementation were measured. Afterwards, a comparison of the extended *IPsec*, the default *DTLS* and optimized *DTLS* implementation was performed, as well as a general comparison of *IPsec* and *DTLS* in the area of the *IoT*.

Therefore, all of the objectives of the project have been accomplished, since the already existing *IPsec* implementation was successfully extended and evaluated in terms of energy consumption, end-to-end delay, packet loss and memory footprints. In addition, another method that can be used in *6LoWPAN* to provide end-to-end security was found (*DTLS*) and optimized. Moreover, *DTLS* was investigated in terms of memory footprints and a comparison of *IPsec* and *DTLS* was accomplished.

Towards this end, an extended *IPsec* implementation for *6LoWPAN* was developed during this project. This implementation was extensively evaluated and we demonstrated that it is feasible to efficiently use *IPsec* in order to secure communication between the hosts in the Internet and sensor nodes. In addition, we showed that the existing *DTLS* implementation for *6LoWPAN* cannot be loaded in extremely resource-constrained devices and a comparison of *IPsec* and *DTLS* was conducted.

9.2 Future Work

Future work that needs to be done with regard to this project, is to evaluate the performance of *IPsec* using hardware, i.e real sensor nodes, instead of the *Cooja* simulator. This will provide more accurate results about the performance and the efficiency of the implementation, since it will be examined in a real life scenario. In addition, hardware *AES* implementation should be used in these experiments, instead of the software *AES* and this will improve the efficiency of *IPsec,* since computations are executed faster in hardware than in software.

Furthermore, this implementation uses pre-shared keys, which are sufficient for the secure communication between any hosts of the Internet, but not very flexible. Therefore, an investigation about how viable is to use *IPsec's* Internet Key Exchange (*IKE*) protocol or another protocol based on *IPsec's IKE*, in *6LoWPAN* must be looked into.

Finally, further investigation and optimization of the *DTLS* implementation is needed. The memory requirements of this implementation are much more than the memory of Tmote sky sensor node. Thus, the code should be examined in more detail and it should be written again in a more efficient way in order to reduce both *RAM* and *ROM* usage.

10. References

[1] Q. Wang and I. Balasingham, "An Introduction, Wireless Sensor Networks: Application-Centric Design," in *Wireless Sensor Networks*, Y. . Tan, Ed. InTech, 2010.

[2] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, New York, USA, 1999, pp. 263–270.

[3] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, New York, USA, 2003, pp. 85–98.

[4] J. W. Hui and D. E. Culler, "IP is dead, long live IP for wireless sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, New York, USA, 2008, pp. 15–28.

[5] S. Raza, S. Duquennoy, T. Chung, T. Voigt, U. Roedig, and D. Yazar, "Securing communication in 6lowpan with compressed ipsec," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, Barcelona, Spain, 2011, pp. 1–8.

[6] W. Gu, N. Dutta, S. Chellappan, and X. Bai, "Providing End-to-End Secure Communications in Wireless Sensor Networks," *IEEE Transactions on Network and Service Management*, vol. 8, no. 3, pp. 205–218, Sep. 2011.

[7] "The Contiki OS. The operating System for the Internet of Things." [Online]. Available: http://www.contiki-os.org/. [Accessed: 13-Apr-2012].

[8] S. Choi and H. Cha, "Application-centric networking framework for wireless sensor nodes," in *Mobile and Ubiquitous Systems-Workshops, 2006. 3rd Annual International Conference on*, San Jose, California, 2006, pp. 1–8.

[9] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 networks," RFC 4944.

[10] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. United Kingdom: Wiley, 2010.

[11] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams in 6LoWPAN Networks draft-ietf-6lowpan-hc-13," 2010.

[12] J. Postel, "Internet Protocol," RFC 791, 1981.

[13] H. P. Seton, "Security Features in IPv6," SANS Institute Reading Room . United States, 2003.

[14] S. E. Deering, "Internet protocol, version 6 (IPv6) specification," RFC 2460, 1998.

[15] K. Seo and S. Kent, "Security architecture for the internet protocol," RFC 4301, 2005.

[16] S. Kent, "IP authentication header," RFC 4302, 2005.

[17] S. Kent, “IP encapsulating security payload (ESP),” RFC 4303, 2005.

[18] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, “COOJA/MSPSim: interoperability testing for wireless sensor networks,” in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, ICST, Brussels, Belgium, Belgium, 2009, pp. 27:1–27:7.

[19] “Tmote Sky: Product Description.” [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf. [Accessed: 13-Sep-2012].

[20] S. Raza, T. Chung, S. Duquennoy, T. Voigt, U. Roedig, and D. Yazar, “Securing internet of things with lightweight ipsec,” SICS, Techinical, 2010.

[21] J. Granjal, E. Monteiro, and J. Sá Silva, “Enabling network-layer security on IPv6 wireless sensor networks,” in *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*, Miami, FL, 2010, pp. 1–6.

[22] P. Vladislav and E. Mehmet, “TLS with PSK for Constrained Devices,” Jacobs University Bremen: , 2012.

[23] “contiki-tls - Implementation of TLS for Contiki OS - Google Project Hosting.” [Online]. Available: http://code.google.com/p/contiki-tls/. [Accessed: 13-Sep-2012].

[24] “contiki-dtls - Implementation of DTLS for Contiki OS - Google Project Hosting.” [Online]. Available: http://code.google.com/p/contiki-dtls/. [Accessed: 13-Sep-2012].

[25] D. McGrew and D. Bailey, “AES-CCM Cipher Suites for TLS draft-mcgrew-tls-aes-ccm-02.”

[26] O. Gay, “Software implementation in C of FIPS 198 Keyed-Hash Message Authentication Code HMAC for SHA2.” [Online]. Available: http://www.ouah.org/ogay/hmac/. [Accessed: 14-Sep-2012].

[27] “Atmel AVR: Product Description.” [Online]. Available: http://www.atmel.com/PFResults.aspx#(data:(category:’34864’,type:!(16)),sc:3). [Accessed: 15-Sep-2012].

[28] O. Garcia-Morchon, S. Kumar, T. Heer, S. Keoh, R. Hummen, and K. Wehrle, “Security Challenges in the IP-based Internet of Things,” *Wireless Personal Communications, Special Issue on the Internet of Things and Future Applications*, vol. 61, pp. 527–542, 2011.

[29] C. Kaufman, “Internet Key Exchange (IKEv2) Protocol,” RFC 5282, 2005.

[30] T. Dierks, “The transport layer security (TLS) protocol version 1.2,” RFC 5246, 2008.

[31] E. Rescorla and N. Modadugu, “Datagram transport layer security,” RFC 4347, 2006.

[32] R. Moskowitz, P. Jokela, P. Nikander, and T. Henderson, “Host identity protocol,” RFC 5201, 2008.

[33] R. Moskowitz, P. Jokela, T. Henderson, and T. Heer, “Host Identity Protocol draft-ietf-hip-rfc5201-bis-03,” 2011.

[34] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin, “Protocol for carrying authentication for network access (PANA),” RFC 5191, 2008.

[35] B. Aboba, L. Blunk, J. Vollbrecht, and H. Levkowetz, “Extensible Authentication Protocol,” RFC 3748, 2004.

[36] T. Ylonen and C. Lonvick, “The Secure Shell (SSH) Protocol Architecture,” RFC 4251, 2006.

[37] R. Moskowitz, “HIP Diet EXchange (DEX) draft-moskowitz-hip-rg-dex-02,” 2011.

[38] S. Raza, A. Slabbert, T. Voigt, and K. Landernas, “Security considerations for the wirelesshart protocol,” in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, Mallorca, 2009, pp. 1–8.

[39] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle, “When hart goes wireless: Understanding and implementing the wirelesshart standard,” in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 899–907.

[40] K. Ren, W. Lou, and Y. Zhang, “LEDS: Providing Location- Aware End-to-End Data Security in Wireless Sensor Networks,” in *25th IEEE International Conference on Computer and Communications*, Barcelona, Spain, 2006, pp. 1–12.

[41] H. Alzaid and M. Alfaraj, “MASA: End-to-End Data Security in Sensor Networks Using a Mix of Asymmetric and Symmetric Approaches.,” in *New Technologies, Mobility and Security, 2008. NTMS’08.*, Tangier, 2008, pp. 1–5.

[42] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. Chang Shantz, “Sizzle: A standards-based end-to-end security architecture for the embedded internet,” *Pervasive and Mobile Computing*, vol. 1, no. 4, pp. 425–445, 2005.

[43] “OMA Technical Section - Affiliates - Wireless Application Protocol Downloads.” [Online]. Available: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html. [Accessed: 25-Apr-2012].

[44] Y. W. Z. Alain, “Securing sensors communications in SmarTB.”

[45] Z. Shelby, K. Hartke, C. Borman, and B. Frank, “Constrained Application Protocol (CoAP) draft-ietf-core-coap-08,” 2011.

[46] B. Karp and H. T. Kung, “GPSR: Greedy perimeter stateless routing for wireless networks,” in *Proceedings of the 6th annual international conference on Mobile computing and networking*, New York, USA, 2000, pp. 243–254.

[47] C. Schurgers and M. B. Srivastava, “Energy efficient routing in wireless sensor networks,” in *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, 2001, vol. 1, pp. 357–361.

[48] T. Zia and A. Zomaya, “A security framework for wireless sensor networks,” in *Sensors Applications Symposium, 2006. Proceedings of the 2006 IEEE*, London, 2006, pp. 49–53.

[49] S. Ozdemir, “Concealed data aggregation in heterogeneous sensor networks using privacy homomorphism,” in *Pervasive Services, IEEE International Conference on*, 2007, pp. 165–168.

[50] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik, “Efficient and provably secure aggregation of encrypted data in wireless sensor networks,” *ACM Transactions on Sensor Networks*, vol. 5, no. 3, pp. 1–36, May 2009.

[51] A. S. Poornima and B. B. Amberker, “SEEDA: Secure End-to-End Data Aggregation in Wireless Sensor Networks,” in *7th IEEE International Conference on Wireless And Optical Communications Networks (WOCN)*, Colombo, 2010, pp. 1–15.

[52] T. Winter and P. Thubert, *RPL: Ipv6 Routing Protocol for Low Power and Lossy Networks, Draft-ietf-roll-rpl-11*. 2009.

[53] “Shamus Software. Multiprecision Integer and Rational Arithmetic C/C++ Library.” [Online]. Available: http://www.cs.sunysb.edu/~algorith/implement/shamus/implement.shtml. [Accessed: 14-Sep-2012].

[54] “Wireshark.” [Online]. Available: http://www.wireshark.org/. [Accessed: 14-Sep-2012].

[55] “Texas Instruments.” [Online]. Available: http://www.ti.com/. [Accessed: 14-Sep-2012].

[56] “IPSecHowTo - Community Ubuntu Documentation.” [Online]. Available: https://help.ubuntu.com/community/IPSecHowTo. [Accessed: 14-Sep-2012].

[57] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and F. T. Voigt, “MSPsim-an extensible simulator for MSP430-equipped sensor boards,” in *European Conference on Wireless Sensor Networks (EWSN)*, Delft, Netherlands, 2007.

[58] V. Perelman, J. Schoenwaelder, M. Ersue, and K. Watsen, “Network Configuration Protocol Light (NETCONF Light) draft-schoenw-netconf-light-01.txt.”

[59] P. Eronen and H. Tschofenig, “Pre-Shared Key Ciphersuites for Transport Layer Security (TLS).”

[60] “The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS.” [Online]. Available: http://www.openssl.org/. [Accessed: 14-Sep-2012].

[61] A. Alshamsi and T. Saito, “A technical comparison of IPSec and SSL,” in *19th International Conference on Advanced Information Networking and Applications, 2005. AINA 2005*, 2005, vol. 2, pp. 395 – 398 vol.2.