# Executive Summary

Over the last fifteen years, the adoption and use of the Wireless Sensor Networks (WSNs) has been increased rapidly and the data collected by sensors became an integral part of our everyday life. But the common belief was that the Internet protocols could not be applied in WSNs because of the nature of sensor devices and their constraints. It was believed that the WSN design would benefit by steering away from the abstraction of layered architectures. However, very recently the TCP/IP stack appeared for embedded devices and demonstrated that IPv6-based Sensor Networks (6LowPANs) are viable and in many cases outperform the traditional approaches.

However, there are challenges associated with key management as the process of key exchange in 6LowPANs is not trivial. Currently, the keys are not disseminated to the nodes of the network, but pre-loaded to the devices at the network initialisation phase. The permanent storage of the pre-installed encryption keys is not only important security vulnerability, but also affects the scalability and the performance of the network.

The main scope of the project is to investigate whether the existing key exchange techniques, developed for the traditional WSNs, are applicable and viable for IPv6-based and lossy wireless sensor networks. Additionally, we are also trying to examine how the key exchange process affects the performance and the scalability of the network. In order to fulfil the scope of the project:

- I have extensively studied about the area of the WSNs and particularly for the 6LowPANs, gaining knowledge of the underlying protocols, specifications and technologies (see page 7).
- By performing a research on a number of different key exchange techniques, which are used in the traditional WSN, I decided that the most promising technique is based on public key cryptography and specifically on the use of elliptic curves (p. 18).
- I have implemented the elliptic curve Diffie-Hellman (ECDH) algorithm, with the use of the ContikiECC library [1].
- I have modified the structure of the link layer frame of the Contiki OS in order to support the auxiliary security header (according to the IEEE 802.15.4 standard) and modified the "create" and "parse" functions of the link layer framer (p. 36).
- I have constructed a transparent, always running process (a daemon), which is responsible for negotiating and establishing a secret key when needed (p. 46).
- A simple solution is developed so that every device is able to store the established secret keys as well as to search for these when a key is requested (p. 37).
- A lifetime mechanism is implemented for all the entries of the Access Control List and for the key exchange process. The mechanism handles the case that the lifetime expires and erases the entry. Also, if the ACL if full, it finds the oldest entry and replace it (p. 38).
- A number of different simulation experiments is designed and run to assess the viability of the key exchange solution by examining the time, memory and energy consumption (p. 55).
- I have tested how the lifetime of the keys affects the energy consumption of the devices and the overall performance of the network, by examining the packet-loss ratio (p. 58).
- I have also tested the scalability of the network in relation to the size of the ACL, the number of the established keys and the hop-by-hop characteristics of the network (p. 65).

# Contents

# 1. INTRODUCTION

It is observed that over the last fifteen years, the adoption and use of the wireless sensor networks (WSNs) has been increased rapidly and that the data collected by sensors became an integral part of our everyday life. The WSNs consist of a large number of autonomous devices (sensor nodes) which cooperate to collect important data and send them through wireless communication channels to a base station or a data centre. Every sensor node (mote) mainly consists of a microcontroller, a memory unit, a transceiver, a power source and one or more sensors. Due to the small size of the nodes as well as the nature and cost of hardware, there are constraints on the resources such as the provided amount of memory, energy consumption, bandwidth and the speed of computation.

In the past, the sensor nodes produced from two different manufactures were incompatible, and due to heterogeneity it was not possible to be used in the same wireless sensor network in order to serve the same application. In 2003, the IEEE 802.15.4 standardization body has adopted a specification standard for communication in order to avoid this problem. This standard specifies the physical and the data link layer characteristics for low-power communication. As a result, all the effort was focused on the link layer of the network stack and different protocols were used on top of the IEEE 802.15.4 standard. Thus, the design of WSNs has become application-centric and therefore different networks cannot be connected to interoperate, as they are incompatible.

The common belief in the WSN research community was that Internet protocols could not be applied in WSNs because of the nature of sensor devices and their constraints, and that WSN design would benefit by steering away from the abstraction of layered architectures. However, recently the TCP/IP stack appeared for embedded devices [2] and demonstrated that IPv6-based Sensor Networks are not only viable, but also equal in performance to the existing WSN solutions. Also, in many cases the IPv6-based Sensor Networks can outperform traditional approaches and as a result, the Internet standards have been adopted for the transmission and routing of IPv6 datagrams over low-power and lossy IEEE 802.15.4 networks (6LoWPAN) [3]. The 6LowPAN standard has been based on the IEEE 802.15.4 standard for the physical and link layer specifications, and additionally defines all the characteristics of the upper layers in order to be compatible with the Internet standards. Thus, the 6LowPAN drives towards the fulfilment of the vision of the '*Internet of Things*' [4][5].

The above approach is beneficial, since two or more sensor nodes from different WSNs are able to communicate reliably, while they are still communicating with other devices connected to the Internet (sensor, server, client machine). Moreover, this approach provides scalability as well as interoperability between different sensor networks. It is also remarkable that the infrastructure of the Internet is already available and its technology is now in a mature and reliable state. Consequently, there is currently a variety of useful tools, available for network management, performance monitoring, route tracing and more. On the other hand, an embedded device connected to the Internet, faces considerably more security threats than traditional IEEE 802.15.4 WSN devices, because of the fact that Internet is public and available to everyone. Moreover, it is of great importance to note that such devices have limited resources like memory and processing power.

The modern sensor devices are often equipped with an encryption co-processor, aiming to achieve secure transmission of the network frames without consuming microcontroller (MCU) cycles. Also, the link layer of IEEE 802.15.4 networks (6LowPANs reside on top of the IEEE 802.15.4 protocol) can support encryption of data frames with the 128-bit Advanced Encryption Standard (AES) algorithm. However, there are challenges associated with key management, and the process of key exchange in 6LowPANs is not trivial. Currently, the keys are pre-loaded to the nodes of the network and this can be characterized as important security vulnerability. Actually, there is a gap in our understanding on how encryption keys should be disseminated to the network nodes.

## 1.1 The scope of the project

The scope of this project is to investigate whether the existing techniques (designed for the traditional, application-centric WSNs) for dynamic generation and exchange of cryptographic keys are applicable and can be adopted in 6LoWPAN networks [3]. The techniques which seem to be viable and applicable will be implemented in the Contiki embedded OS [6], in order to test and evaluate their applicability, viability and scalability for large scale IPv6 networks. In addition, the implemented techniques will be compared with the Cooja simulator [7], which is part of the Contiki OS, to assess their performance and the suitability, so that a suitable technique to be adopted . A comparison will be drawn on quantifiable metrics such as the number of MCU cycles, memory requirements, running time and energy consumption.

## 1.2 The structure of the project

In the second chapter, we will briefly present the background of the IEEE 802.15.4 protocol, which outlines the physical and link layer specifications as well as the 6LowPAN network stack. The 6LowPAN protocol specifies the requirements of the sensor networks in order to be compatible with the current standards of the Internet. Moreover, we will investigate the applicability and viability of techniques based on key pre-distribution and the existence of a base station in the sub-network. Here, it must be noticed that the role of the base station can be performed by the sub-network router or by a server located in the sub-network. Additionally we will present and investigate the current key exchange and authentication techniques for traditional WSNs, which are based on public key cryptography (PKC).

The third chapter presents the working environment of the project and the implementation details of the chosen key exchange technique. Specifically, we provide the details of the Elliptic Curves, which are used to implement the Diffie Hellman algorithm (ECDH) as well as the modification we have performed to secure the link layer frames. Also, we develop and present a solution to store the established keys, to search, replace and delete them when the key lifetime expires.

In the forth chapter, we describe the simulation experiments we have performed, to present and analyse the obtained results. We analyze and discuss the different results in order to validate the implemented technique and to extract statistics and conclusions. We do not assess only the characteristics of the ECDH method but also the performance of the entire network in relation to the number of the nodes and the lifetime of the key.

Finally, we draw the conclusions of the current project by critically evaluating the work that is done, giving suggestions about the network and the established keys. Moreover, we suggest the work to be done in the future in order to provide a complete solution for securing the link layer frames of the 6LowPAN networks.

# 2. BACKGROUND

In this chapter we will present in brief the basic background of the project. Initially, we will give an overview of the WSNs, and we will present some details of the underlying protocols. Moreover, we will present the IPv6-based and lossy wireless sensor networks (6LowPAN) [3] and draw a comparison to the traditional WSNs. In the second section of this chapter we will study the current key distribution schemes used in WSNs, which are basically based on the use of pre-loaded keys, the existence of a Key Distribution Centre (KDC) and the probabilistic key pool protocol.

Moreover, we will give the basic background of the Public Key Cryptography (PKC) and introduce the Elliptic Curve Cryptography (ECC). We will study ECC in depth and examine the characteristics of the elliptic curves as its usage appears to be very promising. Finally, we will present the findings of the research and propose the technique which seems applicable for implementation.

## 2.1 The WSN and 6LowPAN

In this section we will try to present the background and some details of the IEEE 802.15.4 standard, defining the specifications of the physical and the MAC layer of the traditional WSNs. Basically, the IEEE 802.15.4 standard [8] has become a *de facto, state-of-art* radio medium for communication in WSNs. Moreover, the specific standard is important for us because it has been adopted as the basic medium and link layer standard for IPv6 low-power and lossy networks (6LowPAN). After that, we will present the design principles of the network transport layers in traditional WSNs and draw a comparison with the IP-based network stack protocols implemented in 6LowPANs. Also, we will discuss interesting aspects which affect the overall performance of the network, such as the use of duty cycle in sensors, and some important security issues and their impact.

### 2.1.1 IEEE 802.15.4 - PHY and MAC standard

The IEEE 802.15.4 standard as it was revised in 2006 [8] is backward compatible with that version of 2003 and defines the main characteristics of a low-range WPAN as:

- Wireless transmission with data rates of 250 kb/s, 100kb/s, 40 kb/s, and 20 kb/s
- Network topologies of star or peer-to-peer (as well as mesh topology)
- Two types of addresses: 16-bit short addresses or 64-bit extended addresses
- Provides multicast and broadcast capabilities to the nodes
- The allocation of guaranteed time slots (GTSs) is optional
- The access to the channel is achieved with the use of Carrier sense multiple access with collision avoidance (CSMA-CA). It also supports slotted CSMA-CA with the use of beacons for synchronization.
- It is an acknowledged protocol, with a "handshake" operation, for transfer reliability
- Low power consumption
- Energy detection (ED)
- Provides 16 channels in the 2450 MHz band, 30 channels in the 915 MHz band, and 3 channels in the 868 MHz band
- The frame size is defined to be up to 127 bytes
- The link layer security is achieved with the use of 128-bits AES.

It stated above that the frame size is up to 127 bytes, but as it can be seen in Figure 2.1 the actual payload is very limited after framing, addressing and defining the fields for optional security. Actually, this is unavoidable and stems from the constraints of the device.

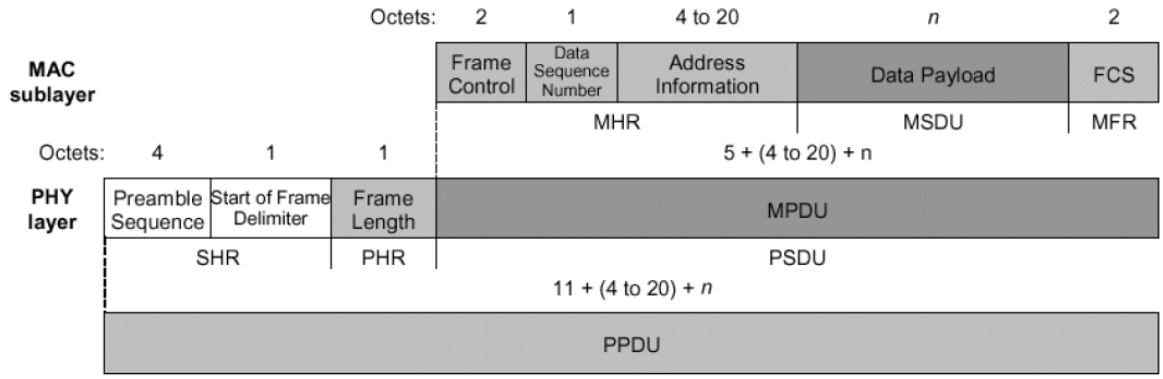


**Figure 2.1**: The data frame as it is defined in the 802.15.4 standard [9].

The overall size of the physical layer frame is 133 octets [10] and as Figure 2.1 shows every frame transmission begins with the preamble sequence; a sequence of 32 "zeros" (4 bytes) is used for access synchronization in the wireless medium, in order to avoid collisions. Moreover, the next two transmitted bytes define the start and the length of the link-layer frame, respectively. The MAC frame is the actual payload of the physical frame (PSDU) and has size up to 127 bytes, where the first two bytes are used for frame control and the next one defines the sequence number of the frame. Moreover, up to 20 bytes are used for addressing, defining the source and the destination MAC address. Furthermore, up to 14 bytes from the data payload are used for encryption and authentication, but the use of security is not mandatory in IEEE 802.15.4 standard, so the specific fields are not set by default. These fields are defined according to the needs of every particular application. Finally, two bytes are used in the end of the frame as checksum (frame check sequence). It is observed that the actual MAC payload has size of 102 bytes if security is not enabled and only 88 bytes when the encryption and authentication mechanisms are in use.

#### 2.1.1.1 Security of IEEE 802.15.4

As specified in the protocol, the need for security is very high as the wireless sensor networks are vulnerable to active and passive attacks, like every other wireless network. This mainly happens because of the wireless nature of the network, where the attacker does not need physical access to the wire to perform an attack. Also, due to the limited capabilities of the devices it is very difficult to ensure that the communication is performed securely.

In order to achieve secure communication the standard specifies that the AES encryption algorithm, which is based in symmetric-key cryptography, must be used. However, it does not provide details on how the keys are created or disseminated. It only mentions that the keys are provided by processes on higher layers and assumes that the keys are stored securely. It is also important to note that the IEEE 802.15.4 standard allows the use of group keys, where a common key is used from a group of nodes (devices) mainly for multicasting and broadcasting. As mentioned in the standard, when a shared group key is used the provided protection is only against the outsider nodes and not against malicious nodes in the specific group, sharing the same key.

According to [11], the MAC security options (fields in the MAC frame) are not enabled by default every time a data frame is sent over the wireless channel. The required security must be defined by the application, which is responsible to define the specific fields. Thus, it is very possible that data are transmitted in plain by fault or bad application design. It is also important that the use of a group key is not recommended [10] and preferably to be avoided because it is very possible that the secret key can be leaked. Actually, every sensor node has an Access Control List (ACL) to store entries for the cryptographic keys and the nonces, which are all associated with a specific destination address. If there are two or more ACL entries with the same key, it is possible that the same nonce will be used again by fault (as the nonce is created from the sender's address, the frame counter and the key counter). In this case the attacker is able to get the secret key by simply applying the XOR operation to the two encrypted messages. It must be noticed that every sensor node can register a group shared-key as multiple ACL entries.

Additionally, the standard defines 8 different security suites for authentication, encryption, and for both. As reported in [10], if a suite which is not providing any authentication mechanism is chosen, the destination sensor is vulnerable to certain attacks, such as denial of service. The attacker can easily trick a sensor node by spoofing the sender address and increasing the recipient's replay counter to the maximum allowable value. As a result the sensor node will become unable to accept any message from the original sender.

It is proposed in [12] that the security in WSNs should be implemented in the link layer in order to avoid the excessive consumption of energy and bandwidth resources, when an adversary is trying to inject routed packets in the network. Furthermore, they draw a comparison on some symmetric encryption algorithms and propose that the most suitable encryption algorithms are AES and XTEA. This is because they need the least time to perform encryption, decryption and the key expansion operations (key expansion is the transformation of the key internally in order to be used in the next round of the algorithm, usually by performing the XOR operation). Also, according to [12], both AES and XTEA offer the best throughput when a key of 128-bits size is used. The comparison of the operation time of the symmetric algorithms is presented in Figure 2.2 and the throughput in the Table 2.1. These algorithms have been evaluated on the MSB-430 platform, which is a Modular Sensor Board hardware platform by the ScatterWeb [13].

In the current project we propose that the AES-128 algorithm should be adopted in the future for symmetric encryption, after the secret symmetric key is distributed. The use of AES was not decided only because it is proposed in the IEEE 802.15.4 standard but also because of its performance as shown in Figure 2.2 and Table 2.1.

| Symmetric Algorithm | Key Length | Throughput (Kbps) |
|---|---|---|
| AES | 128 | 56.39 |
| Skipjack | 80 | 145.45 |
| 3DES | 168 | 12.01 |
| XTEA | 128 | 59.26 |
| RC5 | 128 | 29.49 |
| Twofish | 128 | 10.31 |

**Table 2.1**: The throughput of symmetric encryption algorithms according to the size of the key [12]

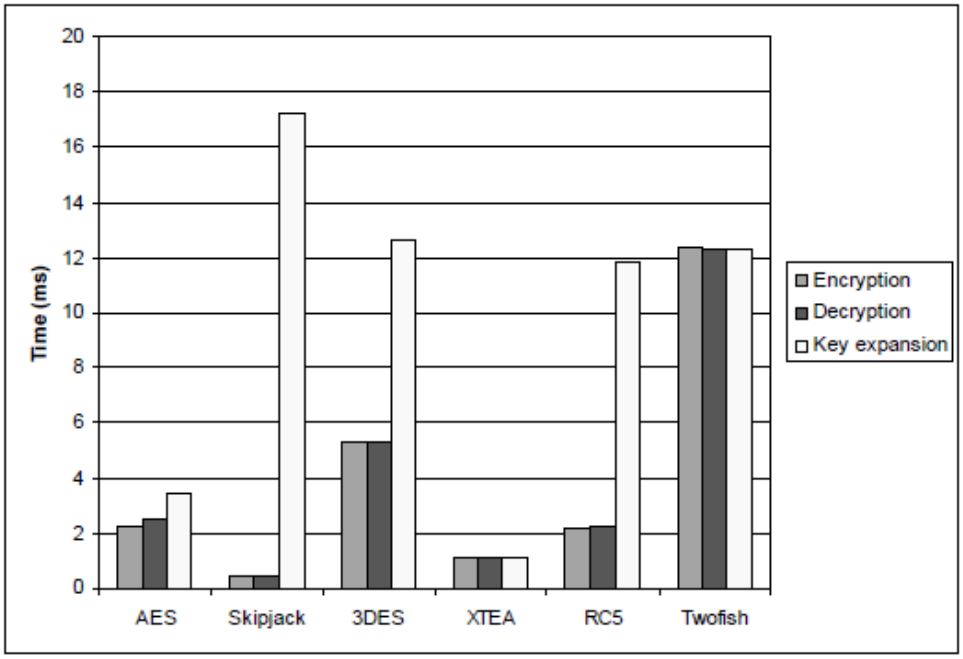


**Figure 2.2**: Encryption, decryption and key expansion time for symmetric algorithms [12].

### 2.1.2 Application-centric network design

The network design in the traditional WSNs is mainly led by the aim of every application in order to achieve specific functionalities. It is believed that the application-centric network design increases the efficiency of the source code and can achieve operating overhead [14] in comparison to strictly layered designs. In such designs it is very common to remove the boundaries between the network layers and to allow the application layer having the responsibility for the particular functionalities, task management and performance monitoring.

The reason leading to the application-centric design is mainly that the applications are data-driven. At most times, the sensed data are related to a specific phenomenon or an event and not collected in regular basis, but only when the event occurs. Also, in many cases we only care for localized data and data collected by a small portion of the available sensors, in relation to the network topology. There are also many monitoring applications (most for military) where an object, human or vehicle is tracked continuously and all the collected data are sent for real time processing.

A major problem stemming from the application-centric design in the WSN area is that the existing routing protocols are not applicable. Consequently, new protocols are designed to perform the operation of data routing between the nodes and the sink (base station). But, as mentioned earlier, the design of every particular network does not follow the layered architecture neither the design of other similar WSNs and therefore new algorithms are needed. The large number of the current routing protocols for WSNs, which are oriented to serve specific applications, is shown in Figure 2.3.

The above problem is not limited only to routing protocols, but also stands for transport layer protocols as their design is influenced from the different routing modes. As expected, new transport protocols are designed with specific characteristics (they do not follow the classic transport mode of the layered architecture) to serve different applications and routing requirements. Because of the nature of WSNs and the hardware constraints, it is not easy to design a new scalable transport protocol to handle end-to-end communication and performing "handshaking". The "handshaking"

procedure uses acknowledgments which result to higher costs in terms of memory, bandwidth and processing power. Most of the current transport protocols for WSNs intend in the exchange of UDP-type packets instead of TCP-type packets, as there is no need for acknowledgments and end-to-end "reliable" communication. Since many WSN solutions are localized and event-oriented, the data traffic is very high when an event occurs [14] and thus it is necessary to use transport mechanisms to control the congestion.

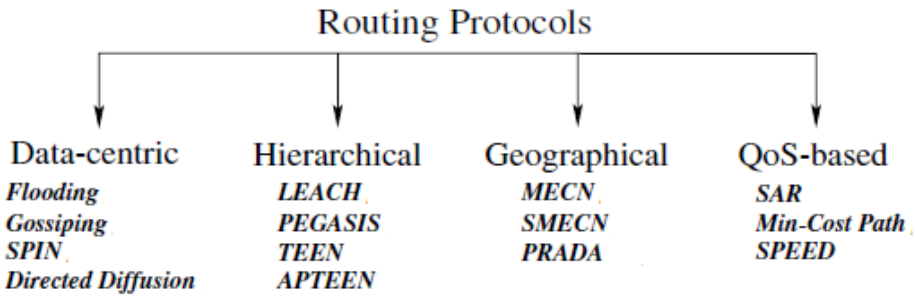


**Figure 2.3**: Overview of routing protocols for WSNs [14]

Usually, the WSN design follows the IEEE 802.15.4 standard for physical and link layer specifications and all the other protocols are built on top of the link layer (Medium Access Control - MAC). As mentioned earlier, the design is application-centric and there are no distinct boundaries between the upper network layers. There are also some cases where the design does not even follow the IEEE 802.15.4 standard neither the upper network stack and we end up to *cross-layer* solutions, which are guided absolutely from the aim of the specific application.

It can easily be observed that the key management schemes in WSNs are influenced in both cases from the network design and that most of the existing key exchange techniques were built to serve application-centric designs according to specific requirements.

### 2.1.3 6LowPAN - Layered network design

In this section we will give a brief description of the 6LowPAN architecture and an outline about the network design by explaining the basic concepts and presenting its main functionalities. If we understand all these concepts we will be able to identify the similarities and the differences to the traditional WSN design and their performed operations. Only in this way we can infer whether a key exchange technique used in WSNs is applicable to 6LowPANs.

#### 2.1.3.1 6LowPAN Architecture

A low-power wireless area network (LowPAN) is an autonomous IPv6 wireless network and consists of a large number of wireless embedded devices, where every device is acting as a node or as an internal router. As shown in Figure 2.4, a LowPAN can be a simple IPv6 network connected to the Internet through an *edge router* [15] or an extended network consisting of a very large number of embedded devices, using multiple edge routers. It is important to mention that the LowPANs are *stub networks* [15]. More specifically, every network device is able to send packets to destinations out of the LowPAN through the edge router or receive packets from the Internet, but does not

forward received packets to other networks; the LowPAN is not a transit network. Also, it is possible that a LowPAN is not connected to the Internet, where it stands as an ad-hoc network and uses the IP network stack only for internal routing between the nodes (there is no edge router).

Another important aspect is that all the nodes of the LowPAN and the edge router share the same 64-bits of the IPv6 address (common IP address prefix) and thus, the address of a node does not depend on its location in the network. In the case of extended LowPANs, there are multiple edge routers connecting the extended network to other IP networks. Not only the nodes that share the same IP address prefix are connected to other IP networks as previously mentioned, but also all the edge routers. Exactly, the IP address prefix is advertised from the edge router; it is given to every internal router and in turn every router distributes the prefix to the nodes, while the Neighbor Discovery (ND) process is taking place.

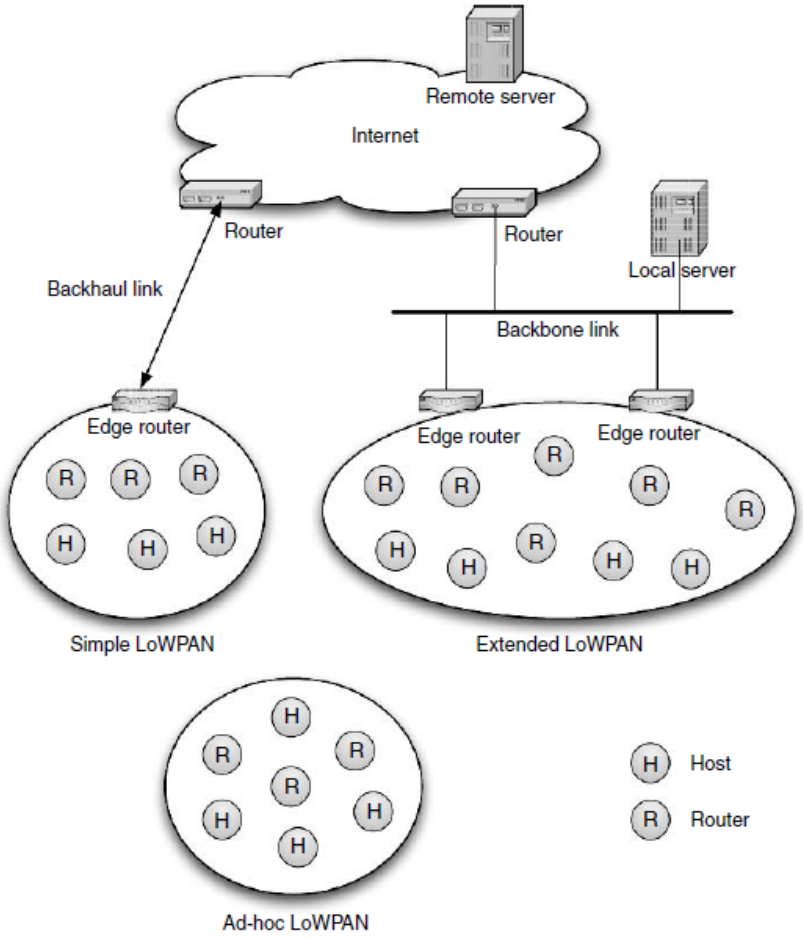


**Figure 2.4**: The LowPAN architecture [15]

The edge router is the most important part of the network as it is responsible for achieving the communication of the nodes with the external networks by routing the incoming and outgoing packets. Moreover, the edge router handles the compression of the IP packets and transforms them in order to be compatible for routing in IPv4 networks. Another important operation of the edge router is that it performs the Neighbor Discovery process, so that every node of the network is registered with the specific edge router. This results to the definition of the interactions between the nodes and the routers.

Furthermore, the 6LowPAN nodes are provided with mobility, not only throughout their specific LowPAN but also between the edge routers in an extended network and even between different low-power wireless networks. The 6LowPAN can support two techniques to achieve communication between the nodes: the first relies on the link layer by forwarding link layer frames and the second on the network layer, where IP packets are routed to reach the destination (like classic routing in the Internet). When a 6LowPAN node wishes to establish communication with a node outside of its network (e.g. a node in different LowPAN, a server in the Internet), it sends IPv6 packets to the edge router that it is registered to, and in turn the edge router is responsible to route the packets to the destination (the edge router performs operations as fragmentation and reassembly of the packets when it is necessary, as well as packet compression).

#### 2.1.3.2 6LowPAN - Network stack

As mentioned earlier, the IEEE 802.15.4 standard defines the specifications of the WSNs for the physical and link layer. Moreover, the IEEE 802.15.4 standard has been adopted as the baseline for 6LowPANs, but also a LowPAN adaptation layer is used as part of the link layer to provide specific features. Actually, the LowPAN adaptation layer is responsible for ensuring compatibility with Internet protocols and optimizing the use of IPv6 packets over the IEEE 802.15.4.

The link layer must be able to provide framing and addressing to support 6LowPANs. It is essential that the link layer supports the use of unique addresses in order to distinguish the link-layer transmission from IPv6 transmission in the case where the IP addresses are compressed and thus are omitted. Also, the IPv6 requires that the packet size, more specifically the *maximum transmission unit* (MTU) for a link, is at most 1280 bytes. This can be achieved in the adaptation layer where fragmentation of the packets is performed.

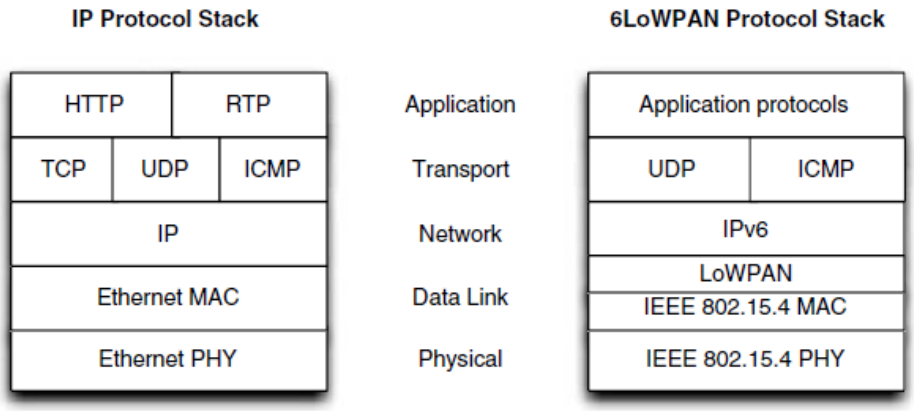


**Figure 2.5**: The IP protocol stack and the 6LowPAN protocol stack [15]

The implementation of the 6LowPAN network stack is almost identical to the classic Internet stack with only few differences as shown in Figure 2.5. A basic difference is that the 6LowPAN stack supports only IPv6 and not IPv4 packets. If necessary, the transformation from IPv6 to IPv4 and vice versa is performed in the adaption layer of the network's edge router. Also, as 6LowPAN uses the IP protocol, it could possibly use the IPsec suite for secure communication, which can guarantee authenticity and confidentiality between the communicating nodes. According to [16] IPsec is not suitable for 6LowPAN because of the embedded devices' limitations on processing and power resources, which in effect leaves sensor devices with the inability to operate all the IPsec algorithms.

Also, IPsec consumes valuable bandwidth by generating redundant packets as it requires a large number of signalling messages [17]. Consequently, it is recommended that the encryption and authentication is provided by the link layer as the proposed key exchange scheme must not generate packets of large size and provide communication overhead. Although, it is reported in [18] that IPsec is viable for 6LowPANs, but further research is needed.

As mentioned above, the 6LowPAN network layer supports only IPv6 in contrast to the classic IP protocol stack, which supports both IPv6 and IPv4. Also, it is important that the transport layer of 6LowPANs uses at most the *User Datagram Protocol* (UDP) for communication because the provided overhead is limited and the UDP can be compressed to the LowPAN format. The "classic" *Transmission Control Protocol* (TCP) is not preferred in 6LowPANs as it provides low performance and high communication overhead; encompassing the capabilities of these devices (the nodes have to communicate multiple times to establish a secure channel by "handshaking"). Also, as the 6lowPAN protocol stack is identical and compatible to the Internet protocol stack, it supports the *Internet Control Message Protocol* (ICMP). The ICMP is mainly used in the Neighbor Discovery (ND) process and similarly to the Internet for control messaging. The protocols used in the application layer of the 6lowPAN protocol stack mostly depend on the specific applications.

The 6lowPAN network stack is divided into layers and is almost identical to the Internet protocol stack. In contrast, the network stack for traditional wireless sensor networks does not follow the layered architecture as mentioned earlier. While traditional WSNs mostly follow cross-layer solutions, we do not know whether a key exchange technique for WSNs is suitable for 6lowPANs. Also, it is possible that a technique which seems applicable for 6LowPAN is not providing the same performance and energy consumption as in WSNs.

### 2.1.4 Radio duty cycle

Sensors are battery supplied and usually are left to act unattended in environments where it is not easy or not possible for us to perform management and control operations. Because of energy constraints the devices are duty cycling their radio to minimize energy consumption, as most of the energy is consumed on idle listening of the channel. Practically, the nodes are in a "sleep" mode for most of the time and periodically "wake" to receive incoming data packets. If a packet is transmitted in the wireless medium while the receiver node is duty cycling, the packet will not be delivered; it will be dropped and retransmitted. However, every retransmission costs because the transmitter node consumes energy, the wireless medium is busy and bandwidth is spent. It is always a trade-off between the duty cycle time and the number of lost packets, so the optimal solution is to maximize the duty cycle period and minimize retransmissions.

There have been various proposed solutions for reducing the number of lost frames but most of them are application-centric as they follow the design of the specific WSNs. These refer to the specific MAC protocols as discussed in [19]. In [20] they introduce an announcement layer on top of the link layer, which is responsible to coordinate the communication by broadcasting beacons. The nodes are able to push data to their neighbors and also to pull data when they "wake up". A duty cycle aware solution for broadcasting is presented in [21], but it seems that it is not useful for 6lowPAN, where packets are routed to reach a specific node. In [22] the idea of probabilistic forwarding is presented because, as they mention, the nodes are not awake at the same time.

As 6LowPAN adopts the IEEE 802.15.4 standard for link layer specifications, we must take in account the sensor's duty cycle period and check which key exchange technique can be adopted. Probably, the performance of the proposed technique will be affected from duty cycle, so we have to specify its impact on the network and evaluate its performance.

## 2.2 Key management schemes for WSNs

The key management schemes proposed for WSNs are divided mainly into two different categories. The first category consists of symmetric key schemes where the keys are pre-installed or assigned by a trusted party and the second category consists of schemes based on public key cryptography. In this section we will focus on the symmetric key exchange techniques as public-key techniques are discussed later, in section 2.3. The field of key management schemes and key exchange techniques for traditional WSNs is very mature, especially the category of pre-shared keys. A simple search in a scientific database provides at least 400 related scholarly articles written in the last ten years [23]. For this reason, we will try to divide the first category into general frameworks and subgroups, and examine the functionality according to the frameworks.

Generally, most of the WSNs use symmetric key management schemes, which rely on pre-shared keys between the nodes or the base station to achieve secure communication. This mainly happens because the required microcontroller cycles and computation time are less than in other schemes (public-key schemes). Most commonly, the keys are pre-installed during the network's deployment and initialization phase. Then, in the network formation phase, the two neighbor nodes discover a shared key and the establishment of the key is performed [24]. But, the approaches presented in this section require human involvement to pre-install the keys as well as taking actions for management and performance monitoring. Also, taking into consideration the capabilities and constraints of the nodes, it is very possible that a node under attack leaks its stored pre-distributed keys and thus the network becomes vulnerable.

### 2.2.1 Key Distribution Centre (KDC) - Trusted entity

The first approach for distribution of a secret common key between two nodes requires the existence of a trusted party in the WSN, acting as a Key Distribution Centre (KDC). The role of the KDC can be taken by the Base Station (BS) of the WSN or by a server destined only to disseminate keys, such as Kerberos [25]. As a matter of fact, Kerberos was designed to provide a mechanism for authentication between two entities and not for key exchange, so the key is delivered as a "side effect" [26]. If node *A* wishes to communicate securely with node *B,* it sends a message to *B* containing its identity, asking for communication. Then, node *B* sends a message to the KDC (containing its own and A's identity) asking for a secret key. After that, the KDC disseminates to both *A* and *B* a common secret key for symmetric encryption.

In the current scheme, the KDC must have secure communication channels with every node of the network in order to deliver the secret keys. While the communication is performed wirelessly and directly (not by routing) the KDC must have unique secret pairwise keys with all the nodes of the network (pre-shared keys, as described in section 2.2.2). Moreover, in the above scheme the KDC must be located within the transmission range of the node asking for a symmetric key (assuming that the transmission range of the KDC includes all the nodes).

**Conclusions**

This solution seems not to be applicable for implementation in 6LowPAN as they are large-scale networks and the transmission range of the nodes is very limited. Also, the communication in 6LowPANs is not only performed directly between the nodes (because of the range), but also the messages (packets) are routed to reach the destination (multi-hop communication). The trusted entity-based key exchange scheme cannot be used as stand-alone scheme in 6LowPANs but possibly it can be used in conjunction with other techniques (key-pool scheme, public-key cryptography) to improve the performance of the network and reduce the communication overhead.

### 2.2.2 Pre-shared keys

In the second framework we categorise all the approaches where the keys are pre-installed in the nodes (offline instalment) and are used directly for communication. Specifically, these keys are not used to create, negotiate, and establish a secret session key; instead they are used directly for symmetric encryption and communication. In the first case, all the nodes use the same shared key and thus the required memory is very limited. This approach is very scalable as a new node can be easily added in the network, but it does not provide enough security guarantees and the entire network is vulnerable if the key is leaked. The second case is that each node has a pre-installed pairwise key with each one of the other sensor nodes; such an extreme scheme is presented in [27]. This scheme provides perfect resilience as the network is not affected if a key is leaked. Specifically, if one of the keys is leaked the communication in the network is not affected, but only the specific pair of nodes cannot communicate securely. The main drawback is that every node consumes a large amount of memory to store the keys. Also, the network is not scalable as for a node addition we need to import a new key to every node in the network. There is a variety of different schemes and specific cases in this subgroup, such as the case of a group of nodes sharing the same key, or a node acting as a server shares the same key with every node in the network.

The main advantage of all the approaches in this framework is that the computational overhead for secure communication is very limited. But, they are all providing low overall resilience [23] as well as low scalability. The first reason is that this approach affects the scalability of the networks in that every sensor element must store a large number of keys, where most of them are probably not used. Moreover, it is very hard, expensive and complex to manage all nodes' keys in order to add or remove a node to/from the network, as well as to perform re-keying operations.

**Conclusions**

As mentioned previously, the 6LowPAN networks require high scalability as new nodes can be assigned to the network dynamically by the network administrator or autonomously, as they have mobility capabilities. It is also important that the nodes in 6LowPANs are provided with high security level as they are connected to the public Internet and exposed to many threats. For the above reasons the use of pre-installed keys for encryption and communication is not recommended.

### 2.2.3 Probabilistic key pool scheme

The probabilistic key pool approach is characterised as one of the best approaches ever proposed for use in WSNs. It was firstly presented in [28] for use in large scale distributed sensor networks, trying to eliminate the disadvantages of the schemes with pre-shared keys (section 2.2.2). In large scale distributed networks it is infeasible to store all the keys for every pair of nodes in each device because of their limited memory. The basic idea behind the key pool approach is quite simple; every sensor node does not have all the available secret keys pre-installed in memory, but only a small distinct subset [28], which is chosen randomly out of a large key pool. The pool is created before the deployment phase of the network and every node is assigned a subset of keys. It must be noticed that every key in the pool as well as every sensor node of the network has their unique identification numbers. Moreover, the scheme works with the existence of some controller nodes, which have more capabilities in terms of wireless range and storage capacity. Every controller shares a distinct key with every node in its range, in order to be able to perform and manage operations such as key revocation and re-keying.

After the deployment of the network, every node exchanges the identifiers of its keys with its neighbors. If they have at least one common key, it is used as their pairwise key for symmetric

encryption and communication. Also, the sensor nodes have the restriction not to use the same key for communication with multiple nodes in their range as the key must be unique to provide the required security.

As a matter of fact, the size of the pool is very large in comparison to the number of keys every sensor node is assigned to ([28] mentions that the pool contains 100.000 keys and every node is assigned a subgroup of 250 keys). Because of the probabilistic nature of the aforementioned key management scheme, many pairs of distinct nodes do not share a common secret key and thus they are unable to communicate (for subgroups of 75 keys the probability for a common secret between two nodes is 0.5). To eliminate this problem, when the network is setup all the nodes are exchanging their key identifiers with their neighbor nodes within their wireless range. If two nodes do not have a common secret key, they are trying to find a secure routing path through their neighbors in order to communicate for key negotiation and establishment.

The connectivity of the network is analysed with the theory of random graphs. The network is represented as the graph G(n, p), where *n* is the number of the nodes and *p* is the probability that two nodes of the network have a link (a secret key). Given the probability $P_c$ for the desired connectivity, the function p can be expressed as:

$$P_c = \lim_{n\to\infty} P_r\left[G(n,p)\ is\ connected\right] = e^{e^{-c}},\ p = \frac{\ln(n)}{n} + \frac{c}{n}, \tag{2.1}$$

where *n* is the number of nodes in the network and *c* is a real constant. By knowing the number of nodes *n* and the probability $P_c$ we are able to calculate the average number of edges each node has, such that $d = p * (n-1)$. Moreover, the desired probability of the network connectivity can be expressed in relation to the size of the key pool *R* and the number of keys *k* every node stores. So, the probability that two nodes of the network share at least one key is given by:

$$p = 1 - \frac{((R-k)!)^2}{(R-2k)!R!} = 1 - \frac{\left(1-\frac{k}{p}\right)^{2(R-k+0.5)}}{\left(1-\frac{2k}{R}\right)^{(R-2k+0.5)}} \tag{2.2}$$

The network described above, operating with the key pool scheme, is very scalable as easily new nodes can be installed. Firstly, we must install a unique key to every new node, for communication with its controller, and then we must assign to these a chosen subset of random keys from the pool. After that, the node can be employed to the network to negotiate and exchange keys with its neighbors. It is also important to examine the above scheme from the scope of the provided security. Actually, it does not provide resilience as the nodes are unshielded and vulnerable to capture attacks. Because of the limited capabilities of the node elements it is easy for an attacker to capture a node and get its group of keys or erase them from the node's memory. If the keys are erased the specific node is not able to communicate with others, so it cannot operate and provide services. If the keys of a specific node become known to the attacker, the other nodes of the network are also vulnerable because the leaked keys belong to the pool and are also used by other nodes.

Different schemes, which modify the basic one, have been proposed to equip the network with resilience. Two such schemes are proposed in [27], where the former recommends that the secret key must not be established only from one common key, but that multiple common keys must be used as well as their hashing string. By this approach, the performance of the network is affected as the communicating nodes must have more than one keys to be able to communicate and because it provides communication and computation overhead. Also, in [24] they express the opinion that the

network will be vulnerable with this approach if the number of nodes is large. The latter approach mentions that it is better to use multiple paths (communication through the neighbors) to establish the communication link and not just to broadcast it to the recipient node. Similarly to the former one, it provides communication overhead as multiple nodes are involved in the negotiation and multiple transmissions are taking place.

The nodes of these networks (based on key pool schemes) are not fully connected to each other because of the probabilistic nature of the framework. If the size of the pool is much larger than the size of the key-subsets, there is always a chance that a pair of nodes does not have a common key for communication. Also, if the connectivity of the network graph is not high, it is possible that the network becomes partitioned into two sub-networks and thus it is impossible to discover paths for key establishment between the two parts. In [23], it is mentioned that the networks must be fully connected to be used in the context of the *Internet of Things* (IoT) as every sensor node must be able to communicate with all the other nodes (nodes acting as servers). Consequently, the pool-based schemes are not recommended.

It is important that the nodes of the WSNs, which use pool-based schemes, communicate directly after the establishment of the key. Because of this fact, the communication is achievable only if the two participating nodes are within each other's transmission range. It seems that communication depends on the location of the nodes and thus the nodes in these schemes are not capable to be provided with mobility. In contrast, as mentioned previously, it is possible to have large-scale extended and interconnected 6LowPANs where the communication does not depend on the sensor's location. The communication in 6LowPANs can be accomplished by transmitting directly to the destination or by routing the frames to intermediate nodes (routers). Also, the nodes in such networks are able to move, according to the 6LowPAN specifications.

**Conclusions**

The basic schemes in the key pool framework seem to be unsuitable for 6LowPAN networks as they are probabilistic, not providing guarantees for the connectivity of the network and the direct communication links between two distinct nodes. Moreover, it was mentioned that networks using the key-pool scheme are very scalable as new nodes can easily be added to the network. However, every node of the network needs to store all the established keys permanently in memory, which is a drawback affecting the scalability of the network. The 6LowPANs are large scale networks but at the same time, the sensor devices used by the network are memory and energy constraint. Moreover, there is not a mechanism to revoke a secret key and establish a new one, but if such mechanism would be existent, the number of different keys that two specific nodes are able to create is very small. This happens as the secret keys depend on the common keys contained in the key-pool of the nodes.

Moreover, the 6LowPAN networks support the multi-hop communication and the mobility of the sensor nodes in contrast to the traditional WSNs. On the other hand, the key-pool protocol does not allow two nodes to communicate if the first one is not in the range of the second, as routing is not performed. As a result from the above, the sensor nodes are not able to move through the large-scale network, establishing single-hop links with their neighbors and by routing achieve hop-by-hop communication with the others.

## 2.3. Public Key Cryptography (PKC)

In this section, we will present and discuss some public-key cryptography algorithms in order to achieve dissemination of a secret shared key between two nodes in WSNs. We will present the

classic Diffie-Hellman algorithm and how authentication of the nodes is achieved with the use of the station-to-station (STS) protocol and the digital signatures. Also, we will examine the RSA encryption algorithm and discuss how a secret key is exchanged. Moreover, we will present the elliptic curve cryptography (ECC) and the Diffie-Hellman scheme over elliptic curves (ECDH) as well as the elliptic curve digital signature algorithm (ECDSA) for authentication.

In Table 2.2 we present the recommended minimum size of the keys for each of the discussed algorithms according to the National Institute of Standards and Technology (NIST) [29]. The recommended sizes for security strength of 80-bits (symmetric) are still secure, but probably would not be in the next few years.

| Algorithm security lifetimes | Symmetric key strength (bits) | DH - DSA | RSA | ECC |
|---|---|---|---|---|
| 2010 | 80 | 1024 | 1024 | 160 |
| Through 2030 | 112 | 2048 | 2048 | 224 |
| Beyond 2030 | 128 | 3072 | 3072 | 256 |

**Table 2.2**: Recommended minimum size of keys by NIST

### 2.3.1 RSA encryption algorithm

We can achieve secure communication between two nodes with the use of RSA encryption algorithm, as proposed in [30]. In brief, if node *A* wants to communicate with node *B,* it constructs a pair of keys (public and private key) and sends its public key to *B* over the unsecure channel of the network. Also, *B* generates a pair of keys and sends its public key to *A* (the keys are not generated every time, instead are reused). Then, *B* encrypts its message with *A*'s public key and sends the encrypted message to *A*. Similarly, *A* encrypts its outgoing messages with *B*'s public key. Those encrypted messages can only be decrypted with the private key matching the specific public key used for encryption. So, the encrypted messages cannot be decrypted by other malicious nodes except from the real recipient, as they are travelling through the network.

To make it clear, we will provide a brief background on how the keys are constructed and how they are used to perform the encryption and decryption of the messages. First of all, it must be noticed that the RSA algorithm is secure because it relies on the *factorisation problem*, where it is very difficult to find the two prime factors of a large given number.

Node *A* (and every node of the network) chooses two random prime numbers $p$ and $q$ where $p \neq q$ and calculates the modulus $n$ such as $n = p \cdot q$. Moreover, it calculates the $\phi=(p-1)\cdot(q-1)$ and chooses the public exponent $e$ so that $e$ is a positive number, relatively prime to $\phi$ and smaller than $\phi$. After that, the node computes its private exponent $d$, which is unique, as $d= e^{-1} \bmod \phi$. At this point, the public key ($K_{pub}$) of the node consists of $\{e, n\}$ and the private key ($K_{pri}$) of $\{d, n\}$, where the public key is sent over the network and the private key is stored and kept secret. After the creation of the keys the prime numbers $p$ and $q$ are never used again and preferably, they are thrown away.

Moreover, the plaintext $m$ can be encrypted with the public key of the node $\{e, n\}$ and the ciphertext $c$ is computed by $c = m^e \bmod n$, so that the ciphertext can travel through the network securely. On the other hand, the encrypted message (ciphertext) can be decrypted only with the use of the stored private key by computing the $m = c^d \bmod n$. From all the above, it is obvious that the attacker can

decrypt all the encrypted messages only if he knows the private key of the recipient node or if he has succeeded in getting the prime numbers *p* and *q* by factoring the modulus *n.* So, the RSA algorithm is more secure as the key size is getting larger; currently the acceptable length of the key is at least 1024 bits and the recommended size is 2048 bits.

It is reported in [31][32][33] that the RSA encryption algorithm is viable in WSNs despite the fact that it uses a large key. It is also recommended by the NIST that a specific pair of keys must not be used for a long time, but both keys must be replaced by a new pair regularly (at least once a year). Therefore, the process of key generation is very slow and energy consuming and additionally, every new generated public key must be certified from the Certification Authority (CA). It is also important that every time we produce a new pair of RSA keys, we must choose new random primes *p* and *q* in order to have different modulus *n*. It is crucial to generate "really" random prime numbers *p* and *q*. However, most of the sensor devices, due to their constraints, are not equipped with "truly" random generators and usually the created prime numbers depend on previous values or pseudo-random algorithms.

As mentioned earlier, the 128-bit AES algorithm is specified in the IEEE 802.15.4 standard and most of the sensor devices are capable to perform AES encryption by having an encryption co-processor. Also, the AES algorithm performs the encryption process much faster than RSA so it would be a good solution to use RSA only to exchange the 128-bit shared secret key, which will be used for symmetric encryption.

Another important fact is that the RSA algorithm can also be utilised for authentication by the communicating entities. Actually, after the creation of the private and public key, each node can encrypt a message with its private key in order to create a digital signature. The digital signature is attached to the data (to the original message - plaintext) and the data is sent over the network. On the other hand, the recipient node separates the ciphertext (signature) from the plaintext (original message) and uses the sender's public key, which is publicly known, to decrypt the ciphertext and compare it with the received plaintext. This operation is known as the verification of the signature. At this point, it must be noticed that the message is encrypted with sender's private key at first, for verification, and then with the receiver's public key to be transmitted securely, as mentioned earlier. The receiver node at first decrypts the encrypted data with its private key and then by using the sender's public key verifies the authenticity of the message, by comparing the signature to the plaintext.

The above digital signature scheme, which uses RSA keys, does not only verify the authenticity of the sender entity, but also ensures the integrity of the transmitted message. The signature is created with the originator's private key so if an attacker tries to modify the message the signature will not be valid; it would not match the original message after the decryption and thus the destination node will reject the message.

In practice, the digital signature scheme is not used exactly as described above because we prefer the signature not to have the same length as the original message. This is mainly because:

- In many cases the original message has very large size and thus the transmitted data over the network will be doubled, consuming bandwidth and memory.
- The encryption and decryption with public key methods are slow processes, consuming processor's cycles and computational power. It is necessary to avoid the consumption of too much energy, especially in the case of the low-power embedded devices.
- If an attacker gains access to multiple plaintexts and to their associated ciphertexts, he is able to guess the encryption key by employing some cryptanalysis methods, such as the known-ciphertext attack.

Thus, the most common approach is to create the digital signature by encrypting the hash digest of the message. The sender agrees with the recipient on the use of a message digest function and encrypts (signs) the digest of the message to create its signature. Then, it attaches the signature to the plaintext and sends the resulting message over the network. The recipient uses the same message digest function to extract the digest of the plaintext, decrypts the signature with the originator's public key and checks if the two digests are matching. By this way, the signature has significantly smaller size than the plaintext.

**Conclusions**

The RSA algorithm is a viable solution for key exchange and authentication, both in WSNs and 6LowPANs, despite the large size of its keys. But, in comparison to other public key techniques (ECC), as shown in section 2.3.4.1, it takes more computation time, especially for the decryption operation. Also, it consumes very large amount of energy in contrast to the ECC provided energy consumption and thus it is not preferred for adoption neither in WSNs nor in 6LowPANs.

### 2.3.2 Diffie-Hellman key exchange

The Diffie-Hellman (DH) algorithm does not use encryption, like the RSA algorithm, to establish a secret key between two nodes; it is not even an encryption algorithm. Instead, the Diffie-Hellman algorithm is based on the *discrete logarithm problem* (DLP) and uses public-key technology not to distribute the shared symmetric key but to create it. Thus, the communication for key establishment is performed by sending "clear" messages (plaintext) over the network.

First of all, the two nodes agree on the use of a large prime number *p* and a generator *g,* which are specifically chosen carefully, so that for every positive number *n* smaller than *p,* there is a power *k* satisfying the $n = g^k \bmod p$. After that, they both create their private keys, which in any case remain secret and their public keys that are derived from the private keys. A's private key is the random value *x* and its public key is calculated as the value $g^x \bmod p$. Similarly B's private key is the random value *y* and its public key is $g^y \bmod p$. At this point, they both send their public keys over the network (not encrypted) to reach the destination on the other side. After receiving each other's public key they combine it with their private key to create a common shared secret key; *A* then calculates the secret key by $(g^y)^x \bmod p$ and *B* by $(g^x)^y \bmod p$, where:

$$\begin{aligned} &((g^y) \bmod p)^x \bmod p = (g^y)^x \bmod p = g^{yx} \bmod p = \\ &= g^{xy} \bmod p = (g^x)^y \bmod p = ((g^x) \bmod p)^y \bmod p \end{aligned} \tag{2.3}$$

Therefore, without knowing each other's private key they manage to establish a secret value as their shared secret symmetric key for secure communication. This key can be used by a symmetric encryption algorithm, like AES. The Diffie-Hellman algorithm is based on the DLP, where given the value $(g^a \bmod b)$ it is very difficult to guess the value *a* (secret key) even if the values *g* and *b* are publicly known. Consequently, the algorithm is secure as long as there is no fast way to guess the exponentiation and solve the discrete logarithm problem. Currently, the recommended size of the modulus *p* is chosen to be 2048 bits and the provided security level is similar to the RSA algorithm with 2048-bits private key.

The nodes taking part in the key agreement process with Diffie-Hellman algorithm are not able to be mutually authenticated, if they do not use any other authentication method. Thus, the DH algorithm

is vulnerable to the man-in-the-middle attack. The Station-To-Station protocol (STS) [34] is an authenticated key agreement protocol with key confirmation, based on DH and digital signatures, eliminating the aforementioned vulnerability. Each participating node does not only create a private and public key for DH exchange (DH keys) but also has a pair of asymmetric keys (private and public key) as in the RSA algorithm, where the private key is used to sign a signature and the public key for verification.

Very briefly, as in the classic Diffie-Hellman algorithm, node *A* generates its random private key *x,* computes its public key $g^x$ and sends it over the network to node *B*. Also, *B* creates its private key *y* and its public key $g^y$ in the same way and after receiving *A*'s public key $g^x$, it generates the common symmetric secret key K = $(g^x)^y$. The difference with the original Diffie-Hellman algorithm is that at this point, *B* concatenates its public DH key and A's public DH key ($g^y$, $g^x$) and signs these with his private asymmetric key in order to create a signature. Then, it encrypts the signature with the secret key *K* and sends both the encrypted (signed) signature and its public DH key ($g^y$) to *A*. Moreover, after receiving the message from *B, A* computes the secret shared symmetric key K by combining its private DH key (*x)* with *B*'s public DH key ($g^y$). With the key *K*, it decrypts *B*'s signature (which was encrypted by *B* with the key *K*) and verifies its validity, by using B's public asymmetric key. Also, *A* must be authenticated by *B*, so that *A* similarly creates its signature by concatenating the two DH public keys, sings the signature with its private asymmetric signing key, encrypts it with the shared secret key *K* and sends it to *B*. At this point, *B* authenticates *A* by decrypting the message with secret *K* and validating the signature with *A*'s public asymmetric key.

At the end of this process, the two participants are mutually authenticated and they have a common shared key for symmetric encryption and thus secure communication. Also, it must be noticed that authentication can be achieved even in the case they do not know each other's public asymmetric key. In this case, every node sends its public authentication key and a certificate to the other participant and thus the key is certified because the certificate is generated from a trusted authority (CA). The use of public key certificates is achievable in 6LowPANs due to the fact that every 6LowPAN node has a unique IP address and is connected to the Internet and therefore, it can easily get a certificate from a certification authority.

**Conclusions**
As mentioned above, the key establishment with the Diffie-Hellman algorithm is secure if the modulus *p* has at least size of 1024 bits, providing similar security level to the key exchange with asymmetric RSA encryption. However, the classical DH algorithm does not authenticate the nodes, which are taking part, so it was proposed to use the STS protocol. In reality, the station-to-station protocol proposes the use of two pairs of keys by every node (DH key pair to establish the shared secret and a pair of asymmetric keys for signing and verifying the signature). The STS protocol is very "heavy" to be used by embedded devices, where the memory and the energy are very limited and thus, it is almost never preferred for key exchange and authentication in its classical form.

### 2.3.3 Elliptic curve cryptography

Elliptic Curve Cryptography (ECC) is a very promising public key cryptography scheme to be used in resource-constraint environments such as WSNs and 6LowPANs. Taking in account that the provided memory of embedded devices is very limited and that the consumption of energy must be minimized, ECC seems to be a suitable approach for usage. This is because the required key for ECC is very small in comparison to the size of the keys in other traditional public key cryptography (PKC) schemes.

### 2.3.3.1 Background on elliptic curves

The term elliptic curve refers to a form of the Weierstrass equation and particularly, to the "smooth" equations [35]. For simplicity, we will examine and present the elliptic curves over the set *R* of real numbers at the beginning. Most of the elliptic curves (over the prime field) are described by the simplified Weierstrass equation:

$$y^2 = x^3 + ax + b \tag{2.4}$$

A graphical representation of the proposed elliptic curves is presented in Figure 2.6 (there are two elliptic curves created with different parameters *a* and *b*). Elliptic curves consist of an infinite number of elements (points) *P* with coordinates ($x_P$, $y_P$) where $x_P$ and $y_P$ satisfy the above equation and a special point *O* (is called point at infinite). Also, it must be noticed that every point *P* of the curve, with coordinates ($x_P$, $y_P$), has its negative point *-P*. The point *-P* is therefore the reflection of *P* in the x-axis and has coordinates ($x_P$, $-y_P$).

It is very important that the elliptic curves are additive groups; this means that we can perform such operations as the addition of two elements of the curve as well as point duplication and the resulting point is also satisfying the equation of the curve. By combining these two operations, as it will be explained later, the multiplication of a point of the curve with a scalar multiplier is achieved.

As the group is additive, any two distinct points on an elliptic curve *E*, P($x_P$,$y_P$) and Q($x_Q$,$y_Q$), where the point $P \neq -Q$ can be added ($P + Q = R$) to create a new point R($x_R$,$y_R$), which is also in the set of the points satisfying the elliptic curve equation. Examining the addition by the geometry approach, it can be represented by drawing a straight line that connects the two points P and Q, as shown in Figure 2.6(a). The specific line intersects the curve *E* exactly once at the point $-R(x_R,-y_R)$, which is the reflection of the point $R(x_R,y_R)$. Moreover, if the point $P(x_P,y_P)$ is added to the point $Q(x_Q,y_Q)$ and the latter is equal to *P*'s negative point $-P(x_P,-y_P)$, such as $Q(x_Q,y_Q) = -P(x_P,-y_P)$, the result is the *O* point at the infinite. As seen in Figure 2.6(b), the line connecting *P* and *-P* does not intersect the elliptic curve at any third point. Also, by considering that $P(x_P,y_P) = Q(x_Q,y_Q)$ and performing point addition, the resulting point *R* is the doubled *P* point, $R = P + Q = P + P = 2P$.

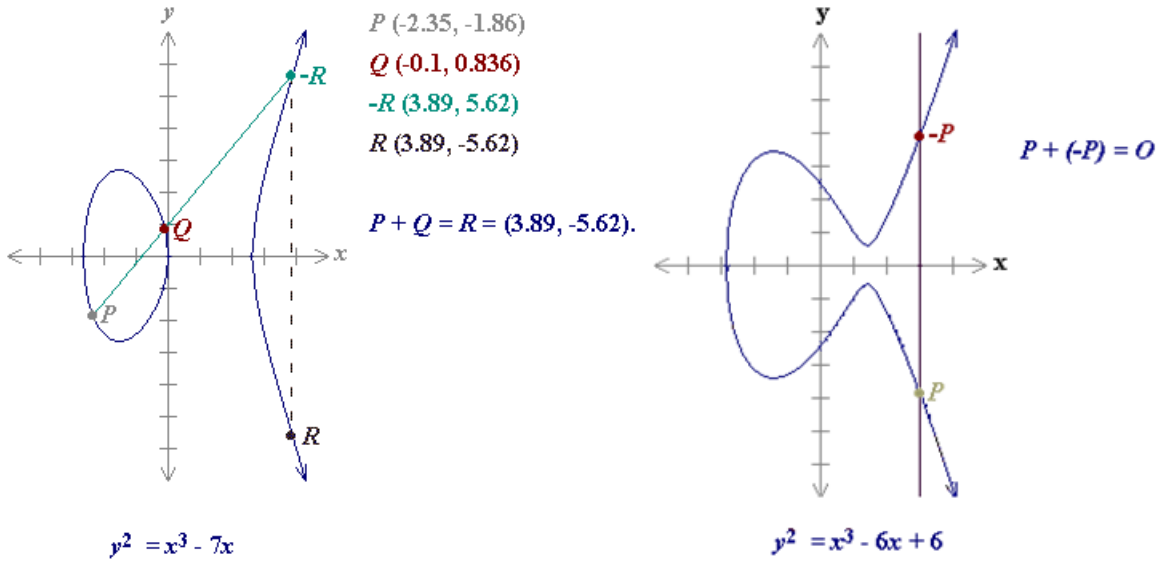


**Figure 2.6**: Point addition in elliptic curves (a) $P + Q = R$, and (b) $P + (-P) = O$ [36]

Because of the underlying mathematics, the coordinates ($x_R$,$y_R$) of the resulting point *R* can be calculated as:

$$x_R = \lambda^2 - x_P - x_Q$$
$$y_R = - y_P + \lambda(x_P - x_R) \quad (2.5)$$

where λ is the slope of the line connecting the points *P* and *Q*, and is given by: $\lambda = \frac{y_P - y_Q}{x_P - x_Q}$ .

Moreover, as mentioned earlier, we are able to find a point *R* on the elliptic curve, which is equal to the double of the point *P,* such that *R = 2P.* Returning to the geometry approach, this can be achieved by drawing the tangent line to the elliptic curve at the point *P*. If the coordinate $y_P$ of the point P is not equal to zero ($y_P \neq 0$) the tangent line intersects the curve exactly once at a distinct point -*R*, which is the negative (reflection on x-axis) of the point *R*. The coordinates ($x_R$,$y_R$) of *R* after performing the point doubling operation on *P* are calculated as:

$$x_R = \lambda^2 - 2x_P$$
$$y_R = - y_P + \lambda(x_P - x_R) \quad (2.6)$$

where λ is the slope of the tangent line on the point *P*, given by $\lambda = \frac{(3{x_P}^2 + a)}{2y_P}$ , and *a* is a real number, a parameter defined by the equation of the elliptic curve.

By repeatedly performing the above operations we are able to multiply a point *P* with any scalar number, depending on the sequence of the performed operations. For example, if we wish to multiply the point *P* by the number four, we calculate the tangent line at *P* to find the point *R* such that *R = 2P* and then doubling the point *R* by calculating its tangent. The point *R'*, where the line intersects the curve, is equals to *2R* and consequently to *4P*. Moreover, by adding points *R'* and *P,* the resulting point is equal to *5P* and if the point *P* is added again, the result stands for the point 6*P*.

The scalar multiplication operation is mainly used to create the public key in ECC systems, because it is very hard to calculate the scalar multiplier and the base point for a given point on the elliptic curve. This is known as the *Elliptic Curve Discrete Logarithm Problem* (ECDLP), solved by many algorithms, such as the Pollard's rho algorithm [37] and the Pohlig-Hellman algorithm [38], but all take exponential time. The fastest algorithm solving ECDLP is the distributed Pollard's rho [39] by using multiple processors, which also takes exponential time. In contrast, the factorisation problem and the discrete logarithm problem (which are related), where RSA and DH relay, can be solved in sub-exponential time by the quadratic sieve algorithm and the number field sieve algorithm [40].

We do not use all the available elliptic curves for elliptic curve cryptography, but only specific finite curves, as shown in Figure 2.7. These curves are finite, containing finite number of points, where every point of the curve has only integer coordinates (not real numbers) and consequently, the rounding error is avoided. The elliptic curves over the finite prime field $\mathbb{F}_p$ (*p* is a prime number) and over the binary field $\mathbb{F}_{2^p}$ are of main interest in cryptography, as they are not vulnerable to sub-exponential attacks [41].

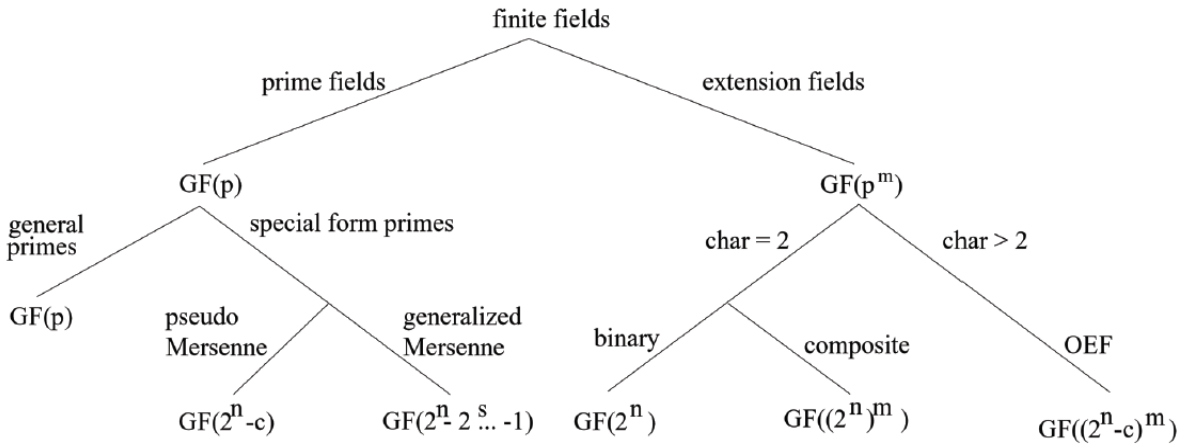


**Figure 2.7**: Finite fields proposed for public-key cryptography [42].

Relating to the above, the point addition operation for an elliptic curve over the field $\mathbb{F}_p$ is calculated by the equations:

$$x_R = (\lambda^2 - x_P - x_Q) \bmod p$$

$$y_R = (-y_P + \lambda(x_P - x_R)) \bmod p \qquad \text{where } \lambda = \left(\frac{y_P - y_Q}{x_P - x_Q}\right) \bmod p \tag{2.7}$$

The point doubling is given by:

$$x_R = (\lambda^2 - 2x_P) \bmod p$$

$$y_R = (-y_P + \lambda(x_P - x_R)) \bmod p \qquad \text{where } \lambda = \left(\frac{(3{x_P}^2 + a)}{2y_P}\right) \bmod p \tag{2.8}$$

#### 2.3.3.2 Elliptic curve Diffie-Hellman

The Elliptic Curve Diffie-Hellman (ECDH) is a public-key cryptographic scheme for key dissemination between two nodes. It is based on the Diffie-Hellman algorithm, but the difference to the classic DH is that the private and the public keys are created with elliptic curve cryptography.

Initially, the two parties agree on the use of a specific elliptic curve over the prime field, by defining the parameters *a* and *b* of the curve, the prime modulus *p,* which has order *n,* and a point generator *G*. Then, each party chooses its own private key *d* so that *d* is a randomly chosen number and smaller than the order *n* of the prime modulus. After that, node *A* takes the generated point *P* of the curve and performs a scalar multiplication of the point, where its private key is used as the multiplier ($Q_A = d_A P$). The multiplication is achieved by performing point additions and point doublings. The point *Q* is used as the node's public key and is sent without encryption over the network. Because of the ECDLP the private key *d* cannot be extracted from *Q* in a reasonable time, so it is considered as secure. The same operation is performed by node *B* in order to create its own public key $Q_B$, by multiplying its private key $d_B$ with the point *P*, so that $Q_B = d_B P$.

When *B* receives *A*'s public key $Q_A$, it multiplies the public key of *A* with its own private key to create the shared secret $s = d_B Q_A = d_B d_A P$. This multiplication is performed by computing point additions

and doublings, similar to the multiplication of the public key computation. Similarly, *A* creates the shared secret key by multiplying its private key $d_A$ with *B*'s public key $Q_B$. At this point, they hold the same secret key for symmetric encryption:

$$d_B Q_A = d_B d_A P = d_A d_B P = d_A Q_B \tag{2.9}$$

The ECDH does not provide authentication between the participating nodes, likely to the classic Diffie-Hellman and RSA key exchange schemes and thus, is vulnerable to the man-in-the-middle attack. This can be avoided by using the elliptic curve digital signature algorithm (ECDSA). The ECDSA is similar to other digital signature algorithms, but the basic difference is that the used keys are created over elliptic curves with point multiplication.

Very briefly, node *A* has the EC key pair ($d_A$, $Q_A$), and also it has the domain parameters (a, b, FP, q, n, G, h) where *a* and *b* are the parameters of the elliptic curve, FP is the representation of the curve, q is a prime number (q = p) or a power of two ($q = 2^m$). Moreover, G is the base point, which is used for the creation of the public key, and *n* is the order of *G*. To sign a signature on a message *m*, node *A* generates a random number *k* and with scalar multiplication, computes the point *R* with coordinates ($x_R$,$y_R$), such that $R = kG$. Then, it computes the inverse of k ($k^{-1}$ *mod n*) and performs the SHA-1 hashing function to the message *m* to obtain *H* and computes $s = k^{-1} (H + d_A R) \bmod n$. The signature of *A* consists of (s, R) and is sent to *B* along with the message *m*.

On the other hand, node *B* obtains a certificate for *A* from a certificate authority (CA) to be able to verify *A*'s signature. The certificate is an authentic copy of *A*'s domain parameters (a, b, FP, q, n, G, h) and its public key. *B*, after receiving the message *m* and *A*'s signature (*s, R*), computes the hashing *H* of the message and calculates the inverse of *s* ($w = s^{-1} \bmod n$). Then, it creates the scalar multiplier $u_1$ ($u_1 = Hw$) and multiplies it with the base point *G* in order to get the point $P_1$ on the curve ($P_1 = u_1 G$). After that, it creates the multiplier $u_2$, where $u_2 = x_R w$, and by multiplying it with *A*'s public key $Q_A$, it obtains a second point on the elliptic curve $P_2$, such that $P_2 = u_2 Q_A$. Finally, *B* performs point addition of the points $P_1$ and $P_2$ and checks whether the resulting point *R'* has the same coordinates with the received point *R* (*R* point of the signature). If so, the signature of A is valid.

However, if the private key of a sensor node is leaked and the adversary gains knowledge of the private key, he is able to calculate the secret shared keys the specific node establishes. This problem can be solved by having ephemeral private keys or by establishing the secret key with the Elliptic Curve Menezes-Qu-Vanstone (EC-MQV) protocol [43].

#### 2.3.3.3 Elliptic curve MQV

The ECMQV key exchange technique is an improvement of the ECDH method, providing implicit authentication of the sensor nodes. If the MQV technique is used for key exchange, the sensor nodes are still secure in the case their private keys are leaked. Moreover, by providing node authentication it eliminates the vulnerability of the man-in-the-middle attack, where an adversary pretends to be the opposite node and tricks both of them by establishing two "fake" secret keys.

The MQV protocol assumes that every sensor node has two pairs of keys. The first pair consists of static private and public keys (long-term keys) and the second one of ephemeral (short-term) private and public keys [44]. Similar to the ECDH protocol, the private key of the sensor node is a large random number and the public key is computed by multiplying the private key with the base point G of the used elliptic curve. Moreover, in the ECMQV protocol the static key pair is used for long time, while the pair of ephemeral keys has a short lifetime value and new keys are computed every time the protocol runs.

According to the ECMQV protocol, the secret shared key is derived from both the static and the ephemeral public keys of the participating nodes. At first, the communicating nodes compute and store their static keys. Every time the nodes wish to establish a secret key, they send their static public key to the other participant and they start computing their ephemeral keys. After the computation of the ephemeral keys, they exchange their ephemeral keys as in the classic ECDH method. It is important to notice that we perform the elliptic curve point multiplication operation to compute every new ephemeral public key, which is actually a resource demanding operation.

When a sensor node receives the static public key of the other node, it performs a point multiplication with the received public key and its own static private key. Moreover, after the reception of the ephemeral public key, it multiplies the received ephemeral public key with its ephemeral private key. The secret shared key is then determined by adding the two points we have just computed.

The authentication in the ECMQV protocol relies on the fact that the static keys of the sensor nodes are permanent and do not change during the time. Thus, the static public keys of these nodes can be authenticated from a certification authority, which ensures the identity of the corresponding nodes. Moreover, the current protocol uses a hash function to convert the points of the elliptic curve into numbers, which is also related to the authentication process.

This method requires the existence of an authentication authority to certify the static keys of the nodes, which is not always possible in the WSN and specifically in 6LowPAN networks as the sensor nodes are provided with mobility. Moreover, the ECMQV method requires on average 2.5 point multiplication operations to establish a secret shared key, according to [45]. In contrast to the ECMQV, the ECDH method requires a single multiplication for each key exchange if the private and the public keys of the nodes are stored permanently. If we decide not to store the keys permanently, but to compute a new key pair for each key exchange process, the ECDH method requires two scalar point multiplications. However, if authentication is required the ECMQV method is almost faster than ECDH along with ECDSA.

#### 2.3.3.4 Optimizations over the elliptic curves

According to [46], it is not necessary to use both coordinates for the representation and calculation of the points over the curve, so the ECDH key exchange scheme and the ECDSA are simplified, while only the $x_R$ coordinate of the point R is used. This observation is very important, especially for resource-constraint devices as the required microcontroller cycles are decreased. Moreover, as presented in [47] and shown in Figure 2.8, by performing some optimizations on ECC computations we can minimize the time spent on multiplication, sign and verification and thus, elliptic curve cryptography becomes an attractive solution.

Specifically, the ECC implementation that is presented in Figure 2.8 uses the secp160r1 elliptic curve of size 160 bits. Specifically, this implementation is done in the TinyOS platform [48], but also contains source code directly written in the assembly language. Actually, it was done exclusively for the MICAz motes and thus, the results are presented in Figure 2.8 were obtained by evaluating the ECC on the specific motes. Also, it is important to mention that the presented optimizations are applied consecutively and therefore, their effect is cumulative.

As mentioned, the EC point multiplication operation is the most important and the most demanding operation over the elliptic curves. Actually, the point multiplication is performed by applying a number of point doublings and additions in a sequence. In the original form of the multiplication, the operation follows the double-and-add method, which is similar to the multiply-and-square method in the modular exponentiation operations (like exponentiation for the RSA algorithm). A given

illustration for this method is that if the next bit of the binary represented scalar multiplier is "0", we perform a point doubling. Otherwise, if the bit is "1", we perform a point doubling and a point addition on top. The pseudo-code of the method is given in figure 2.9.

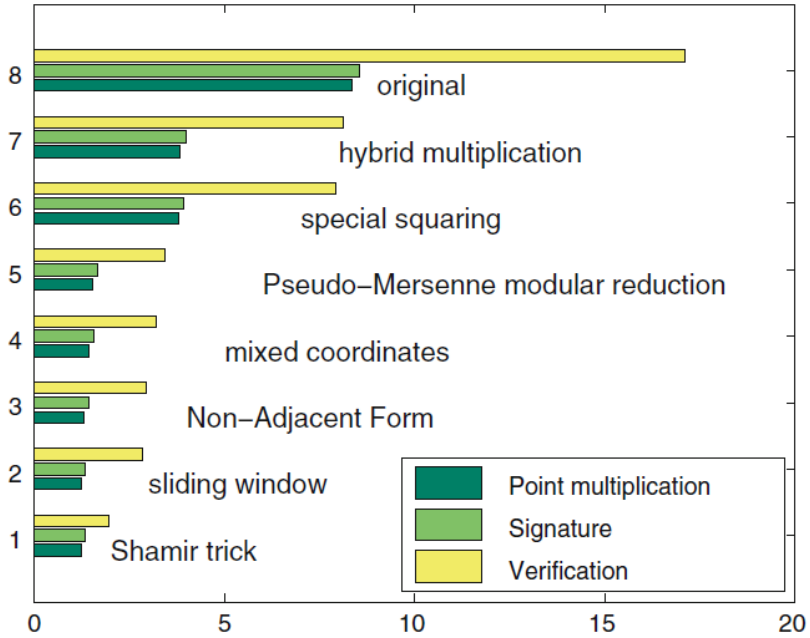


**Figure 2.8**: Computation time for ECC with optimizations [47].

```
* Q = 0
* for i from m to 0 do
    * Q := 2Q (using point doubling)
    * if d_i = 1 then Q := Q + P (using point addition)
* Return Q
```

**Figure 2.9**: Pseudo-code for the double-and-add method for point multiplication [49]

Because of the cost of the original multiplication method, some other optimized methods are used. Specifically, in the windowed method a window of size w is chosen, where w is usually set to the value of four, as the specific size is the optimum according to the NIST. Actually, in the windowed version of the multiplication we pre-compute and store all the $2^w$ points ($d_i = 0, 1, 2, \ldots 2^w-1$) of the curve. Each point in the pre-computed array stands for the multiplication of the specific number $d_i$ with the base point of the curve. With this method we reduce the number of the point additions, where addition is an expensive operation, by having the values pre-computed and stored in an array. Similar to the double-and-add method, we perform a sequence of point doublings, but at most only one point addition for every *w* bits of the multiplier. The pseudo-code of the above method is given in the Figure 2.10.

```
* Q = 0
* for i from 0 to m do
  * Q := 2^W Q (using repeated point doubling)
  * if d_i > 0 then Q := Q + d_i P
    (using a single point addition with the pre-computed value of d_i P)
* Return Q
```

**Figure 2.10**: Pseudo-code for the windowed point multiplication method [49]

An improvement of the windowed method is the sliding-window method, where only half the points are pre-computed and stored in contrast to the window method. Actually, we pre-compute the points corresponding to the most significant bit of *w* ($2^{w-1}$, $2^{w-1}+1$, $2^{w-1}+2$, .., $2^{w}-1$). With this method we not only try to reduce the point additions, but also trade off some additions with doublings, as the doubling operation is roughly faster. Eventually, if we are able to implement and use the sliding window method, there is no need for the windowed method since the latter outperforms the former both in time and energy consumption. The pseudo-code of the sliding window method is presented in Figure 2.11.

```
* Q = 0
* for i from 0 to m do
   * if d_i = 0 then
      * Q := 2Q (point double)
   * else
      * Grab up to w - 1 additional bits from d to store
        into (including d_i) t and decrement i suitably
      * If fewer than w bits were grabbed
         * Perform double-and-add using t
         * Return Q
      * else
         * Q := 2^W Q (repeated point double)
         * Q := Q + tP (point addition)
* Return Q
```

**Figure 2.11**: Pseudo-code for the sliding-window method for point multiplication [49]

Moreover, we can achieve better performance by using the Non-Adjacent Form (NAF) to represent the binary multiplier. In the NAF representation each binary number can be represented as a unique sequence of signed digits, where the constraint is that the representation must not contain two "1" digits in a sequence. This unique representation does not correspond to any other number. Roughly, the operation of point addition over the elliptic curves costs the same as the point subtraction operation and thus, the NAF representation does not provide any additional cost. On the other hand, by using signed digits and by the restriction that a "1" cannot be followed by another, the NAF employs more *zero* digits in the representation of the specific number, in contrast to the binary representation. Precisely, the NAF representation trades off some point addition operations with point doublings, where the addition operation is more expensive than the point doubling.

### 2.3.4 Related work

The key exchange techniques that are based on PKC are characterized as the most basic techniques for key dissemination, both for the conventional wired and wireless networks. Moreover, they are used for a long time now, because of the high level of security they provide. Until recently, it was believed that PKC is impractical and not suitable for wireless sensor networks, because of the network dynamics as well as the provided computational overhead and energy consumption [50]. However, in the past few years, it have been reported that the asymmetric encryption techniques are viable and applicable in WSNs. It seems that the techniques with the best performance are those based on ECC.

#### 2.3.4.1 RSA vs. ECC

It is reported in [31] that they have implemented both RSA and ECC cryptosystems with 1024-bits and 160-bits key size respectively, on MICA mote sensors and that PKC is a feasible security solution for sensor networks. They have reported that the computational time for RSA public-key operations is 4.6s and 389s for private key operations. After using some optimization techniques for large-integer operations they managed to reduce the required time significantly.

In a matter of fact, they have used Montgomery reduction [51] to avoid the division operation, which is expensive, and the achieved time was 1.2s for the public-key operations as well as 82s for the private key operation. Moreover, they have used the Chinese Remainder Theorem (CRT) to reduce both the modulus and the exponentiation, achieving the times of 0.79s and 21.5s respectively. They also present in [31] that the generation of ECC digital signatures takes 1.3s and 2.8s for verification by performing point additions and doublings in mixed Jacobian and Affine coordinates [52], as well as by using pseudo-Mersenne primes to reduce the complexity of the modular reduction operation.

Also in [32], they present an implementation of both RSA and ECC public-key algorithms on the Atmel AVR Atmega128 microcontroller. It is important that the specific microcontroller shares the same characteristics and capabilities with microcontrollers used in 6LowPAN sensor devises. In the RSA implementation they have used Montgomery multiplication and CRT as in [31], but also they have optimized the squaring operation. In the squaring operation for large integers, partial products (some bits) appear twice, so we can avoid recalculating them. The specific optimization reduces computations by up to 25%. Moreover, it is known that memory access is a relatively slow operation in comparison to computation, so they have used row-wise and column-wise multiplications as well as combination of both techniques to optimize memory usage.

The results of [32] for RSA with keys 1024 bits and 2048 as well as ECC with public keys of 160, 192 and 224 bits are shown in Table 2.3. It also presents the amount of memory (in bytes) every algorithm uses, both for data storage and for the code. It is observed that the 160-bits ECC is not only much faster than the equivalent 1024-bits RSA, but it also uses less memory for data and code hosting. The same applies if we draw a comparison between the 192-bits ECC algorithm and the 2048-bits RSA, which provide the same level of security.

The time presented in Table 2.3 is for the elliptic curve multiplication operation and for the RSA exponentiation operation. Actually, in [32] they have implemented the standardised NIST/SECG *GF(p)* curves and performed the optimisations of using the projective coordinate system, the NAF to represent the multiplier and the specific optimisations of NIST/SECG. The implementation of the RSA algorithm uses the CRT and the Montgomery reduction method. Moreover, the implementation was performed on Atmel AVR Atmega128 microprocessor, written directly in the assembly language in order to allow specific memory access optimisations.

| Asymmetric Algorithm | Time (s) | Memory-data (bytes) | Memory-code (bytes) |
|---|---|---|---|
| ECC-160 | 0.81 | 282 | 3682 |
| ECC-192 | 1.24 | 336 | 3979 |
| ECC-224 | 2.19 | 422 | 4812 |
| RSA-1024 public key | 0.43 | 542 | 1073 |
| RSA-1024 private key | 10.99 | 930 | 6292 |
| RSA-2048 public key | 1.94 | 1332 | 2854 |
| RSA-2048 private key | 83.26 | 1853 | 7736 |

**Table 2.3**: Average ECC and RSA execution times and memory consumption

The previous work is extended in [33] as they do not only perform RSA and ECC operations, but also implement RSA key exchange with mutual authentication as well as ECDH with ECDSA, between the two untrusted parties. They provide a simplified and lightweight SSL protocol in order to allow the sensor nodes to perform the "handshake" operation, to negotiate and establish a secret common key. The specific communication protocol follows the client-server model and is very similar to the procedure and the message flow we discussed in sections 2.3.1 and 2.3.3.2. In short, the client node sends a random challenge to the server node and the server responds with a random and its certificate. After the client receives the response, it verifies the server's public key and then encrypts the secret common key and signs it with its private key. After the server receives the message, it decrypts it with its private key and then it verifies it with client's public key and sends the response "finish". The procedure requires only 4 messages to be sent over the network.

The energy costs of the key exchange operation and for signing and verifying the signature on the Atmel Atmega128L microprocessor are given separately in the Table 2.4. It can be observed that the most energy consuming operation is the computation with RSA's private key, which is used both from the client and the server to sign a signature. In the key exchange process only the server needs to use its private key to decrypt the data (the common secret key). On the other hand, the required energy for key exchange with ECC is about the same as RSA public-key operations, both for the server and the client (as they both perform the same scalar multiplications to construct the secret key). The energy consumed for ECC signature verification is more than that of verification with the public key of RSA. If we examine the consumption of energy for a full "handshake" operation and not individually for every module, ECC consumes four times less energy than RSA.

| Algorithm | Key exchange | | Signature sign/verification | |
|---|---|---|---|---|
| | Client (mJ) | Server(mJ) | Sign (mJ) | Verify (mJ) |
| RSA-1024 | 15.4 | 304 | 304 | 11.9 |
| RSA-2048 | 57.2 | 2302.07 | 2302.7 | 53.7 |
| ECC-160 (ECDSA/ECDH) | 23.3 | 23.3 | 22.82 | 45.09 |
| ECC-224 (ECDSA/ECDH) | 60.4 | 60.4 | 61.54 | 121.98 |
| RSA-1024 (Handshake) | 397.7 | 390.3 | | |
| ECC-160 (Handshake) | 93.7 | 93.9 | | |

**Table 2.4**: Energy consumption for digital signatures, key exchange, and a full handshake [33]

However, in all the above related work they do not account the impact of the sensor's duty cycle or they assume that the duty cycle takes only 1% or less of the total time. Also, they do not estimate the average energy consumed for idle listening of the channel in comparison to the energy spent for computation, as well as the energy spent for retransmission of the frames. It is important to check how the duty cycle period affects the performance of the wireless sensor networks and consequently, what is its impact on 6LowPANs. This is an open issue as the impact of duty cycling in 6LowPANs is probably different than in WSNs, because of the fact that frames are routed to reach the destination and are not only transmitted directly.

#### 2.3.4.2 ECC - Identity Based Cryptography (IBC)

In [53], they introduce the asymmetric scheme of Identity Based Cryptography (IBC), which is based on bilinear pairings on elliptic curves, as a promising key exchange scheme for WSNs. In [54], they propose the IBC scheme for IP-based WSNs. The basic idea of IBC is that every string, like the identity of each node (ID), can be used as a valid public key and thus the use of large and "heavy" certificates for authentication is avoided. By this approach the nodes are able to establish a common secret key without any communication. Specifically, if nodes are not already known, they exchange only their IDs and a request for secret key computation. The main advantage of the approach is that the communication overhead in the network and the consumption of energy is minimized.

The proposed scheme is based on bilinear pairings of elliptic curves. A bilinear pairing is a computable mapping of the points P, Q over an elliptic curve having the property:

$$e([a]P, [b]Q) = e(P, [b]Q)^a = e([a]P,Q)^b = e(P,Q)^{ab} \tag{2.10}$$

Moreover, the simplest type of bilinear pairings, used in [53], has the property that $e(P,Q) = e(Q, P)$.

This scheme requires the existence of a Trusted Authority (TA) in the network, responsible to compute and disseminate the private keys of all the nodes. The TA has its private master key S which is unknown to the nodes. All the nodes have a unique identity ID and the hashing of their identity H(ID) as their public key, which is a point on the elliptic curve. In the pre-deployment phase of the network, the TA calculates the private key of each node as S*H(ID) and disseminates them to the nodes, where the keys are stored permanently in the memory. In reality, the private key of these nodes is a scalar multiplication of their public key with the master private key S of the trusted authority. As mentioned earlier, it takes exponential time for a node to calculate S if both H(ID) and S*H(ID) are known because of the ECDLP.

After the deployment phase of the network, if two nodes wish to establish a secret key for secure communication, each multiplies its private key $S \cdot H(ID)$ with the public key of the node on the other side H(ID), so that:

$$\begin{aligned} S \cdot H(ID)_A \cdot H(ID)_B &= S \cdot H(ID)_B \cdot H(ID)_A \\ A_{pri} \cdot B_{pub} &= B_{pri} \cdot A_{pub} \end{aligned} \tag{2.11}$$

The common secret key is established without prior communication of the nodes, as the public keys of all the nodes are based on their identity and are publicly known. Also, a malicious node is not able to guess a pairwise secret key of two communicating nodes from their public keys, as the master S of the authority is secret.

As reported in [53] the above scheme consumes less energy than RSA and ECC to perform a full "handshake" as it does not require authentication. The only messages that are sent over the network are for signalling the beginning and the end of the key computation and thus, the transmitted bytes are much less in comparison to the other public key schemes.

**Conclusion**
In general, the key exchange scheme with bilinear pairings provides good performance and seems to be suitable for the 6LowPANs, since all the nodes have a unique identity (the IPv6 address). But in my opinion, the current technique has a main drawback, which stems from the fact that the private keys are computed only by the trusted authority, disseminated to the nodes and stored locally in the sensor nodes. First of all, if the secret master key S of the TA is leaked, it is essential that the private keys of all the nodes are revoked in order to begin a re-keying phase. Also, while the sensor nodes have limited capabilities, if a node leaks its private key it becomes vulnerable. It is not possible to generate a new private key to replace the old one as all the private keys of the network consist of the node's identity multiplied with TA's secret master key. Consequently, a re-keying phase must take place so that a new master secret S to be chosen.

#### 2.3.4.3 Key exchange in IP-based WSN

It is of great importance that in [55] they have implemented the Secure Socket Layer protocol (SSL) for IP-based wireless sensor networks. They have implemented in the SNAIL platform both the full SSL "handshake" and a lightweight version, reducing the required communication. As it is reported, they use ECC for key exchange (ECDH) and authentication (ECDSA), the RC4 cipher for data encryption and the MD5 and SHA-1 algorithms for hashing (it is used in digital signatures). It is important that the data packets have size of 127 bytes (in accordance to IEEE 802.15.4 standard) and that the adaptation layer of the network stack performs fragmentation and reassemble of the IPv6 frames, according to 6LowPAN standards.

Although, the announced implementation does not prove that ECC key exchange and ECDSA authentication (or the SSL protocol) is applicable for LowPANs because there are two basic differences between the SNAIL platform and 6LowPANs. Firstly, the implementation of the network stack (IP-TCP) in SNAIL is slightly different from the 6LowPAN network stack. Moreover, the sensor devices of SNAIL have more capabilities than 6LowPAN nodes.

#### 2.3.4.4 Clustered network approach

In [56] a protocol for asymmetric encryption with RSA (in a WSN environment) is simulated and the consumed energy "*very optimistically is quite similar to energy requirements*" for symmetric encryption. But, in the scenario of [56] they draw a lot of assumptions to achieve this result and it seems not to be for general purpose. Firstly, they assume that the network is divided into clusters and all the nodes of the cluster communicate with the cluster's head node securely. Moreover, the cluster head node communicates only with the base station of the network and not with other head nodes. Every time a head node wishes to communicate with the base station, they both create and exchange their public keys and therefore, all the transmitted messages are encrypted with their private keys.

Additionally, it is mentioned that they have modified the size of the sent messages to reduce the consumption of energy; an assumption not viable for 6LowPANs, where the size of every frame is defined in the IEEE 802.15.4 standard and modification is not allowed. Due to the assumptions they have assumed, we cannot consider this scenario as a *general* asymmetric key exchange scheme in order to check its applicability for 6LowPANs.

## 2.4. Summary and Conclusions

Currently, we have studied the main approaches used in WSNs for exchange and dissemination of a secret key between the nodes, in order to examine if they are applicable in 6LowPANs. Because of the constraints of the sensor devices, it is recommended that the key establishment as well as the encryption of the transmitted messages should be performed in the link-layer of the network protocol stack. It was argued in many cases that secure channels can be created in the top layers of the protocol stack (network and transport layers) by applying end-to-end techniques used in the classic Internet networks as TLS and SSL. Although, an inherent limitation is the small frame size in WSNs and 6LowPANs (127 bytes, with actual payload from 102 to 88 bytes) preventing us to use the full network and transport layer headers, which are compressed for inner communication.

The basic key exchange techniques are divided into two main categories as we have seen, those using pre-shared keys, stored in the memory of the sensor nodes and those using public-key cryptography to negotiate and establish a secret key. The key management schemes of the first category seem not to be applicable to 6lowPANs due to the fact that they are large scale networks and due to their characteristics, such as the mobility of the nodes and the multi-hop (routed) communication. The first scheme requires the existence of a Key Distribution Centre (KDC), which is reachable by all the nodes in a single hop, to disseminate a pairwise secret key to the nodes that request communication. The second framework requires that all the nodes have pre-installed pairwise keys for communication with every node in the network. This assumption cannot be applied to 6LowPANs because of the large number of the nodes, as every node needs memory to store all the pre-shared keys. Moreover, this scheme does not support the multi-hop communication neither the mobility of the nodes.

The key exchange schemes in the key-pool framework share the same limitations with the previous presented schemes as they do not allow communication for two distant nodes, not being in each other's transmission range. Moreover, the key-pool schemes do not guarantee the connectivity of the network as they rely on the probability of a common key in the node's key ring.

The second category consists of the public key schemes, such as the RSA, the Diffie-Hellman key exchange algorithm, the DH over elliptic curves with authentication and the Identity based key exchange schemes. As it was seen, the IBC schemes consume less energy than the others and provide the minimum communication overhead. But, they have the limitation that a trusted authority is needed to compute the private keys of the nodes and that a re-keying phase must take place if a key is leaked. The classic Diffie-Hellman algorithm is not suitable for WSNs and 6LowPANs as it requires a key of large size and it does not provide any authentication mechanism. In order to achieve authentication, we must employ a digital signature scheme on a later phase. Thus, the provided computation and communication overhead is very high, in addition to the consumption of energy.

Both RSA and ECC (key exchange with ECDH and ECDSA for authentication) are applicable to WSNs and seem to be suitable for adoption in 6LowPANs. The use of ECC is recommended instead of RSA because of the smaller size of the required keys, the provided computation time and the limited consumption of energy. Moreover, the ECC scheme consumes less memory during the computation process and provides minimized communication overhead in comparison to the RSA algorithm.

Because of all the above we have concluded that we will be implementing the ECDH key exchange scheme. We have chosen to use ECC because of the performance it provides for WSNs and the advantages it offers over the other public key schemes. The implementation will then be used to check the viability and applicability of the chosen scheme (ECDH) in 6LowPAN networks as well as to assess the scalability and the performance of the entire network in relation to the key exchange process.

# 3. IMPLEMENTATION OF THE KEY EXCHANGE TECHNIQUE

## 3.1 The working environment

In the current project, we research ways in order to examine if any of the pre-existing key exchange techniques implemented for the WSNs are suitable for IPv6-based, Low Power and Lossy Networks. To accomplish the project it is necessary to use an appropriate platform to implement the most promising key exchange technique (ECDH) and to test its applicability. Moreover, we run a number of different experiments to measure the memory, energy and time consumption of the chosen technique as well as to examine the scalability of the network to conclude whether it is a viable solution.

We have decided to implement the key exchange technique on the Contiki OS, which is a portable and lightweight open source operating system written in standard C language. The Contiki OS is specifically designed for use by devices with limited resources and thus it is widely used in sensor devices, fulfilling the idea of the "Internet of Things". The architecture of the Contiki OS combines an event-driven kernel design together with a pre-emptive multi-threading mechanism, which is called "protothreads" [57].

The Contiki OS supports a full TCP/IP network stack, providing communication according to the standard IPv6 protocol (developed by Cisco) as well as supporting the UDP, TCP and HTTP protocols. It also implements the 6LowPAN adaptation layer as defined in the 6LowPAN standard [3] by the IETF and the RPL IPv6 multi-hop routing protocol for low-power devices [58]. Moreover, the MAC layer implementation of the Contiki OS (the ContikiMAC) [59] allows sensor nodes to stay for the majority of time in a sleep mode. The ContikiMAC reduces the energy spent on idle listening of the radio by the transceiver, by contributing an effective duty cycling mechanism. Usually, the duty cycling time of the devices takes about 99 percent of the total time, even though the communication is achieved with a low to no packet-loss ratio.

In addition, the Contiki OS provides a mechanism for estimating the energy consumption of the system during the running time. Using the Energest mechanism [57] we can not only estimate the overall energy consumption of the sensors, but also the energy spent on individual and specific operations/functions. Furthermore, this mechanism allows the estimation of the energy consumed by different modules of the device separately, such as the microcontroller (MCU), the transmitter (TX) and the receiver (RX).

In order to be able to use the Contiki OS, a virtual machine is employed on my personal computer with a distribution of Ubuntu installed. I have also installed the latest version of Java, as well as the Eclipse IDE for C/C++ development. After that, I configured the Eclipse IDE properly to support the Contiki OS in order to use it for the development of the project and to properly link the files for compilation. Furthermore, I have installed the open-source version control system GIT, which controls and coordinates distributed development. Java, along with the Apache Ant software must necessary be installed in the virtual environment in order to compile and run the Cooja simulator [7].

After the implementation of the key exchange technique, we need to design and build some experiments. These experiments are necessary in order to test and evaluate the implemented technique as well as to examine and analyze the performance of the network. All the experiments have been performed on the Cooja simulator, which is delivered along with the Contiki OS. Actually, the Cooja simulator is an integral part of the Contiki OS as it cannot be used separately to serve a different purpose. In addition, the Cooja simulator is compatible with the Contiki OS and is able to perform simulations of IPv6 sensor networks and carry out different and complex experiments. It provides measurements and results, not only for particular operations of the sensors, but also for

the load and the behaviour of the entire network. The Cooja simulator is a powerful and very useful tool to emulate and evaluate the functionality of the network before the source code is burned into the sensor's hardware. In all of the simulations that we have performed, we utilised the Tmote Sky platform [60] which carries a 16-bit MSP430 microcontroller [61], 48Mb of ROM and up to 10Mb of RAM memory.

## 3.2 Implementation - Methods

In the current project, our main goal is to research for a suitable key exchange technique in order to be adopted in the 6LowPAN networks, securing the link layer frames of the internet protocol stack. The method which as mentioned above seems most viable is the ECDH, mainly because of its simplicity in comparison to the ECMQV method. The main reason we decided to implement the specific key exchange technique at the link layer is that we want to provide security at the lower level of the network. A number of packets are not directly sent to the destination, but are routed autonomously through multiple links, as the communication is multi-hop. The most common reason for a transmitted message to pass through multiple nodes in order to reach its destination is that the two communicating nodes are out of range.

### 3.2.1 Link Layer frame and framer

The existing implementation of the MAC frame in the Contiki OS misses the security header that is specified in the IEEE 802.15.4 standard. Thus, we modified the structure of the MAC frame in order to include and implement the security header. The modification was done in the function of the frame creation as well as in the function parses the received frame. If the application determines that the MAC frame needs to be secure (by defining the security bit in the frame control field), the security header is appended to the address field of the existing MAC header, prior to the data payload as shown in Figure 3.1.

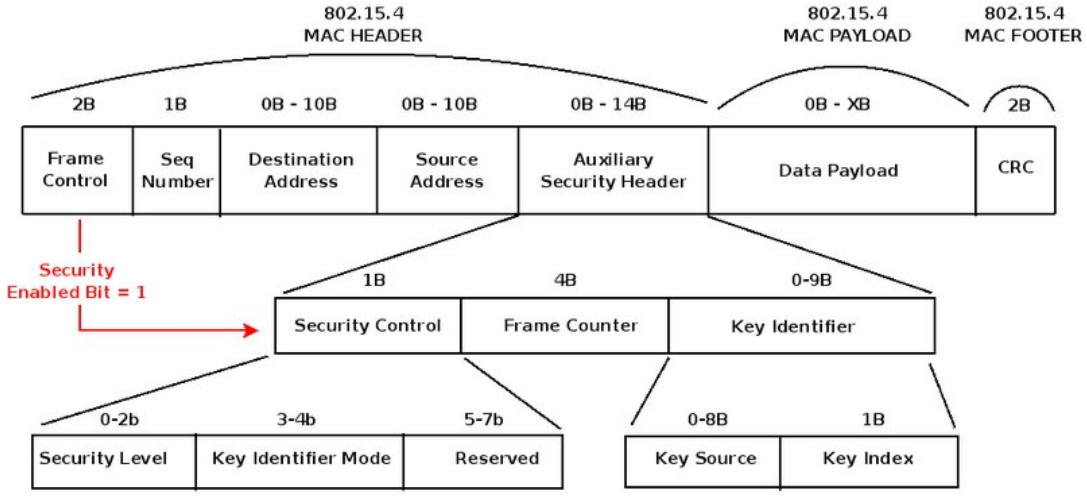


**Figure 3.1**: The MAC Header and the Auxiliary Security Header [62]

The frame creation function is responsible for filling the security header with data, such as the required security level (encryption and/or authentication), the key identifier mode and the security frame counter as shown in figure 3.1. Moreover, if the security level supports encryption and the key is determined implicitly, it adds to the header an index to the specific key. If the key is defined explicitly there must be a feature to include the key in the sent frame. On the other hand, all the

data contained in the security header must be recognised and parsed properly by the receiver node and not be considered as part of the frame payload (data). In this case, the parsing function has to recognize the length of the security header and follow the reverse procedure in order to receive and construct the frame correctly. As mentioned above, we have modified the structure of the MAC header in order to support the security field when is needed and to recognise its existence in the incoming frames.

Except from the structure of the MAC frame, it is also necessary to make some modifications to the MAC framer, which is responsible for creating and parsing every single frame. The framer takes the outgoing packets from the upper layers of the internet stack, adds the MAC header at the beginning of each packet and forwards them to the wireless medium. Also, the MAC framer has the duty to receive the incoming frames, parse them, remove the MAC header and pass the payload to the upper layers. In order to support security, we modified the MAC framer so that it fills the values of the Auxiliary Security Header properly. The values of the auxiliary security header are defined by the application, which determines if security is enabled.

At this point, it must be noticed that the MAC framer is the last component handling the data prior to frame transmission and that it does not take any decision about the destination of the message or the next node the frame has to be forwarded to. The routing decisions are taken by the ContikiRPL [63], which is the implementation of the routing protocol for low-power, lossy networks for the ContikiOS as specified by the ROLL working group [58]. Thus, the destination IP and MAC addresses of every frame are passed in blind to the framer.

To be able to provide link layer security, we implemented a function which takes the MAC address of the destination node, as a parameter, and searches for the secret key assigned to the specific address. If the key exists in the Access Control List (ACL) of the node, it is returned to the framer, otherwise the returned value is NULL. In the case the returned key value is NULL we do not allow the framer to construct the outgoing frame and forward it to the transmitter and therefore, the frame is dropped due to the lack of security guarantees (Figure 3.2). Then, we start an internal transparent process for key exchange, as it will be explained later.

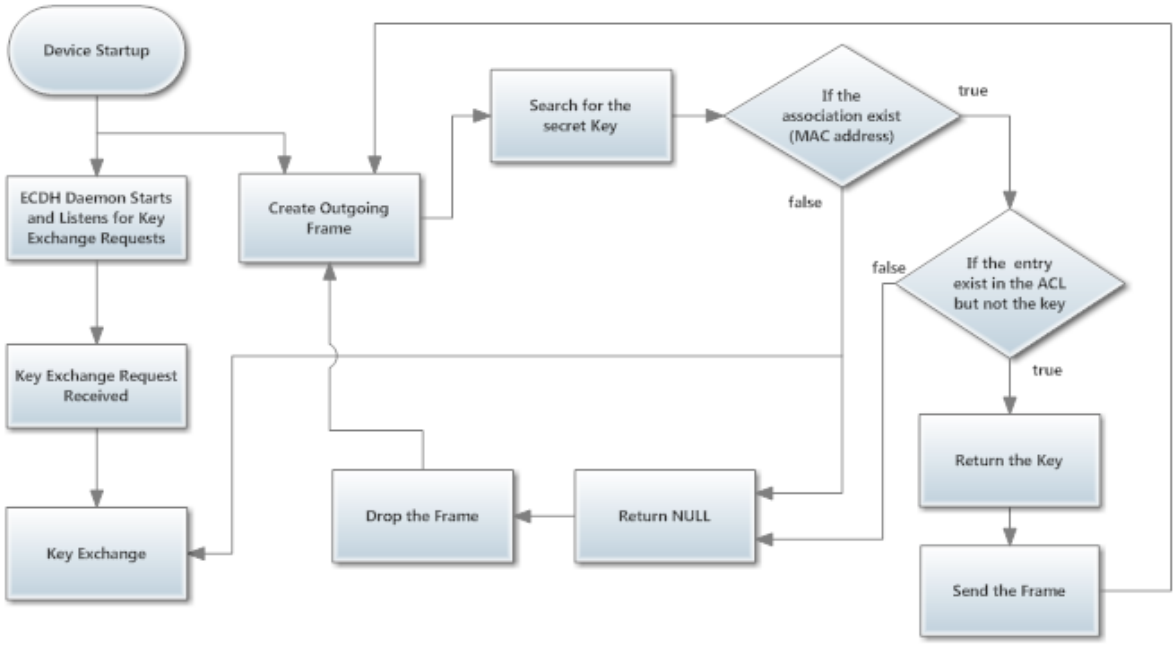


**Figure 3.2**: Flowchart for the frame creation, searching for the key, sending/dropping the frame.

In the case where the secret key exists in the ACL table and is returned, we proceed to the transmission of the outgoing frame. The returned key will normally be used for AES encryption of the outgoing frame, but this is not the main goal of the current project and due to the luck of time we do not apply the encryption scheme at the moment. The main reason for not using the AES algorithm is due to its large memory requirements in order to be implemented as a software solution. Although, we have managed to use and test the functionality of the Texas Instruments' implementation of the AES algorithm [64] as a standalone component, in order to ensure that it works properly. The aforementioned algorithm will probably be used in the future to extend the current work.

An important implementation detail is that the framer asks for the encryption keys only in the case of unicast communication and specifically only if encryption is specified by the application layer, as clearly specified in the IEEE 802.15.4 standard. At the beginning of the project we tried to implement a solution where the framer asks encryption for all the outgoing unicast frames except from these which are responsible for the key exchange process. This decision led to a serious problem as some unicast packets that ensure the functionality of the network were dropped due to the lack of a secret key. Such packets are sent regularly from the sensor nodes to construct the network, keep track of their neighbors and make routing decisions. The effect of the above problem is that the network gradually becomes unconnected until it collapses. To avoid this situation, we took the decision to follow the opposite rule. If the application specifies that security is required, then we allow all the frames to pass except those of the specific application.

Another problem we came across was how to pass the encryption flag (the security bit) from the application layer to the MAC layer, in order to enable the MAC framer to ask for the encryption keys. This problem stems from the principles of the layered architecture and the implementation of the Contiki OS. According to the layered architecture design, the application level data have to travel across all the intermediate layers to reach the link layer, as the direct communication is not allowed.

### 3.2.2 ECC Implementation

In the current project, we used source code from the ContikiECC project [1] in order to implement the basic elliptic curve operations. The ContikiECC project modifies the TinyECC library [65] in order to be ported to the Contiki OS. The TinyECC library is an implementation of the ECC operations, originally developed for the TinyOS [48].

The ContikiECC project implements functions to handle very large numbers as multiple 8-bit or 16-bit words and provides the basic numerical operations on 8-bit and 16-bit microprocessors. These functions can be used not only for elliptic curve cryptography, but also for any other application which requires operations on large numbers. Moreover, the ContikiECC project provides a number of elliptic curves of size 128, 160 and 192 bits (specifies the parameters *a*, *b*) as well as the base point of each curve. In the current project, we have decided to use a curve of size 128 bits in order to construct the secret key, as the sensor hardware is capable to perform the 128-bits AES. The unused elliptic curves are not compiled to avoid unnecessary memory usage. The application developer can easily choose to compile a different curve by defining it at the Makefile.curve_params.

The basic numerical operations for large numbers are used by the functions of the ECC file, in order to implement the basic ECC operations. These operations are the point addition, point doubling and multiplication by a scalar multiplier. The ContikiECC project uses the sliding window method in the implementation of the basic functions, which provides more optimised characteristics in comparison to other methods as shown earlier. The sliding window method consumes a small amount of time prior to the point multiplication operation to calculate and store the points of the window. These points are used to minimize the total number of point additions (as mentioned in section 2.3.3.4).

Moreover, the ContikiECC project uses the above ECC functions (operations) to construct an example of two communicating sensors which use the Elliptic Curve Digital Signatures Algorithm (ECDSA) to achieve mutual authentication. The drawback of the specific example is that it does not employ IPv6/UDP communication to send the signatures over the wireless medium. Also, the source code needed for ECDSA implementation takes significantly large amount of memory in comparison to the code we have used to implement key exchange with ECDH.

Additionally, the ContikiECC project does not provide any method for key exchange, so by using the existing functions for point scalar multiplication, we have managed to implement the Diffie-Hellman method over elliptic curves. First of all, we have created a function, which returns a random number of size 128-bits as a sequence of 16-bit words. This random number is used as the private key of the participating sensor node and remains secret. The next function we have constructed computes the public key of the sensor node by multiplying the private key (the random 128-bit number) with the known base point of the used curve.

For the calculation of the shared secret key, we have constructed another function similar to the previous one for the public key calculation. This function multiplies the random private key of the first node with the public key of the negotiating node after its public key is received. The main difference between the two functions is that the former uses the curve's base point in the scalar multiplication and the latter uses the public key of the other participant. In both cases we pre-compute the values of the sliding window in order to achieve correct multiplication. This happens as the values of the window depend on the specific point of the elliptic curve, which takes part in the multiplication.

The source code used to implement the above functions is taking about half the size of the available code for ECC operations in the ContikiECC project. This effort is an attempt to reduce the necessary used code and keep the ROM memory used in acceptable levels.

### 3.2.3 Storage of the secret keys

As mentioned earlier, the link layer framer calls a function asking for the encryption key only if a unicast frame is set to be sent. The framer passes the specific local address (MAC) of the destination node to the keys function and waits for the secret key. At this point, it is crucial to examine how the mentioned function works as well as how the keys are stored and deleted.

A custom structure has been constructed (called as key association), which contains the MAC address of the destination node, the established secret key assigned to the specific MAC address, the lifetime of the key and the state of the key exchange entry. The information is stored in the structure permanently, but more specifically until the lifetime of the secret key expires. As shown in Figure 3.3, both the private ECC key of the node and the public key of the opponent node, which is received during the key exchange process, are stored in the association entry.

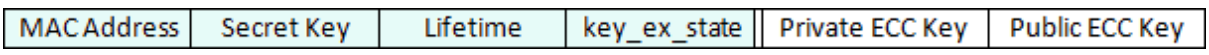


**Figure 3.3**: The key association structure which is an entry in the ACL Table

The reason we store the private key in the ACL table is because the private key of a node is not always the same and is not permanently stored. Every time a node wishes to compute a secret key with a node of the network, a new unique private key is created. Thus, the different private keys are stored to the related key association entries. Not only the private key is stored, but also the received

public key of the other participating node is stored too. This is because we need to temporarily keep the public key in the corresponding entry, in relation to the MAC address. We have decided to store both keys to the related entry in order to allow multiple key exchanges to run concurrently. If we do not manage to store the keys, we need to build a mechanism that prevents multiple key exchanges to run concurrently. If concurrency would not be prevented in the case the keys are not stored, then the key exchanges could affect each other by modifying the intermediate results.

After establishing the secret key, both the private and the public ECC keys of the corresponding association structure are erased. This happens because they are not needed any more as the secret key has been computed and stored. Moreover, by erasing them we prevent the possibility of leakage and we reduce the RAM memory usage.

| | MAC Address | Secret Key | Lifetime | key_ex_state | Private ECC Key | Public ECC Key |
|---|---|---|---|---|---|---|
| 1 | **00:12:74:02..** | **17d3 4af7..** | **455** | **COMPLETE** | NULL | NULL |
| 2 | **00:12:74:03..** | NULL | NULL | **ALLOCATED** | **af33 a760..** | NULL |
| 3 | **00:12:74:04..** | NULL | **42** | **IN_PROGRESS** | **bf01 56a6..** | NULL |
| 4 | **00:12:74:04..** | NULL | **33** | **IN_PROGRESS** | **88c2 ae12..** | **fe45 2188 4a..** |
| 5 | NULL | NULL | NULL | **NONE** | NULL | NULL |

**Table 3.1**: A snapshot of the ACL table of a network node

Every node in the network has an ACL table, which is a statically pre-allocated memory, for storing the association structures of its neighbors as different entries. We have defined the default size of the ACL table to be four entries, but this can easily be changed by defining it in the key exchange header file. Actually, as it will be discussed later, the size of the ACL table determines whether the network is scalable or not. By running an experiment on different ACL sizes we will examine the scalability of the network. An example of the ACL table is given in Table 3.1.

Each key association in the ACL table has a lifetime (measured in seconds) that can be defined by the application layer. If it is not defined by the application, a default value is assigned in the ECDH header file. We will later examine how the key lifetime affects the behaviour of the network, the energy consumption as well as the packet-loss ratio. The lifetime of each secret shared key is set at its creation and is stored in a specific entry of the ACL table.

The internal *purge_association()* function is called periodically with one second time interval and decreases the key lifetime of the valid ACL entries by a second. An entry is valid if the MAC address is not assigned as NULL and the lifetime is greater than the value of zero. Entry one (1) of Table 3.1 is an example of a valid (not expired) association. It can be seen that the specific entry contains the MAC address of the corresponding neighbor, the secret key, a non-zero lifetime and its state is marked as complete. Moreover, it is observed that the private and public ECC key fields are NULL, as the key exchange process is over and a secret key has already been created.

If the lifetime of an association expires, the *purge_association()* function is responsible to remove the association from the ACL table. This is achieved by assigning the MAC address of the entry as NULL and the key state value as KEY_STATE_NONE, as shown in the Code Snippet 3.1. By "erasing" the MAC address, the ACL entry is free to be allocated for a new key establishment in the future. Such free entry is depicted in the ACL association number five (5) of Table 3.1. The role of the *purge_association()* function can be seen in the state transition diagram of Figure 3.5 (top-right corner).

```
static void purge_associations() {
    uint8_t i;
    for (i = 0; i < ECDH_CRYPTO_KEYS; i++) {
        if (!rimeaddr_cmp(((rimeaddr_t *) &associations[i].mac),
                &rimeaddr_null)) {          // if MAC addr is not NULL
            associations[i].lifetime--;     // reduce the lifetime

            // if lifetime is zero, free the entry by setting MAC  as 0
            if (associations[i].lifetime == 0) {
                memset(&associations[i].mac, 0, sizeof(uip_lladdr_t));
                associations[i].key_ex_state = KEY_EX_STATE_NONE;
            }
        }
    }
}
```

**Code Snippet 3.1**: The purge_associations() function.

When the link-layer framer asks for a key, the function *get_key_for_neigh(mac_address)* searches the entire ACL table to match the given MAC address with an address registered in the ACL. If the input MAC address of the neighbor node matches the address of one of the entries in the ACL table and the state value of the key is marked as KEY_STATE_COMPLETE, the key is returned. This means that the two sensor nodes have already established a valid secret key at a prior stage. On the other hand, when the given MAC address does not match any of the registered addresses in the ACL table, the key exchange process starts and the NULL value is returned. Consequently, by returning the NULL value the framer is forced to drop the outgoing frame. This function is shown in Code Snippet 3.2.

```
uint16_t * get_key_for_neigh(uip_lladdr_t * neighbor) {
    uint8_t i;
    for (i = 0; i < ECDH_CRYPTO_KEYS; i++) {
        if (rimeaddr_cmp((rimeaddr_t *) neighbor,
                ((rimeaddr_t *) &associations[i].mac))) {  // the MAC matches
            if((associations[i].key_ex_state == KEY_EX_STATE_COMPLETE)) {
                return associations[i].aes_key;     // the state is complete
            } else {
                return NULL;        // key exchange in progress
            }
        }
    }   // MAC not match, start the key exchange, return NULL key
    start_key_exchange(neighbor);
    return NULL;
}
```

**Code Snippet 3.2**: The *get_key_for_neigh*() function.

However, if the MAC address matches a registered address but the key state is not assigned as complete (allocated or in progress), the value NULL is also returned. Actually, the two sensor nodes are in the process of key establishment, while the process has not finished yet. In this case neither the secret key is returned (the key does not exist) nor the key exchange process begins as the process is already in progress. The overall functionality of the *get_key_for_neigh*() function, as described above, is clearly presented in Figure 3.2, where the framer asks for the secret key at the frame creation time.

We decided to force the framer to drop the frame when the key establishment process is in progress with the specific node (the ACL entry exist but the key is NULL). This is to avoid the situation where

the framer is blocked (and cannot be used by other applications) until the key exchange process is over. The ACL entries corresponding to this situation are the 2, 3 and 4 of Table 3.1.

Concisely, the second (2) entry of Table 3.1 corresponds to the case where an association was free and thus it became allocated and returned. The sender node has assigned the MAC address of the recipient node and its private key. However, the lifetime is still marked as NULL and the key state as ALLOCATED, as the sender node did not send the key exchange request message yet. Actually, this entry reflects the early stage of the key exchange process, where the sender is probably computing its public key.

A very important function that we have implemented is the *allocate_association(mac address)* function, which takes as a parameter a specific MAC address and allocates an association in the ACL table. This function is called internally at the beginning of the key exchange process in order to find and allocate a free association entry. It searches through all the entries of the ACL table to find a free association where the assigned MAC address is NULL. If such an entry exists and the key state value is set to KEY_STATE_NONE, the key state changes to KEY_STATE_ALLOCATED and the association is returned as shown in Code Snippet 3.3 (a).

When the MAC address of the ACL entry is NULL and the key state is not equal to KEY_STATE_NONE, the entry does not become allocated and is not returned. As a result we skip it and we move to the next association in the ACL table, as shown in Code Snippet 3.3 (b). Normally, the MAC address is not NULL if the key exchange process is in progress, but actually there is a small time gap between the association allocation time and the time where the MAC address is assigned to an association. This situation can happen only if the sensor node is at an early stage of the key exchange process with one of its neighbors. By having this control statement, we ensure the correctness of the allocation function, avoiding re-allocation and return of an already allocated association.

```
for (i = 0; i < ECDH_CRYPTO_KEYS; i++) {
    if (rimeaddr_cmp(((rimeaddr_t *) &associations[i].mac.addr), &rimeaddr_null)){
        if(associations[i].key_ex_state == KEY_EX_STATE_NONE){
            associations[i].key_ex_state = KEY_EX_STATE_ALLOCATED;
            return &associations[i];
        }
        else{
            continue;
        }
    }
    else{
```

(a)

(b)

**Code Snippet 3.3**: *allocate_association()* function - Allocation of a free ACL entry

It is also important to examine what happens in the matter of a sensor node wishing to establish a secret key with one of its neighbors, but its ACL table is full of non-expired entries. As previously mentioned, each node searches for a non-allocated entry in the ACL table (with MAC address assigned to NULL) by calling the *allocate_association(mac address)* function. To overcome this problem, we set a temporary lifetime value equal to the maximum available lifetime at the beginning of the process. After that, for every key association with a valid MAC address, we compare its lifetime to the "oldest" temporary lifetime value. If the lifetime of the entry is less than the temporary lifetime, we keep this association as the oldest one and we update the lifetime value of the comparing parameter, as shown in Code Snippet 3.4.

```
            continue;
        }
    }
    else{
        if((associations[i].key_ex_state != KEY_EX_STATE_ALLOCATED) &&
                associations[i].key_ex_state != KEY_EX_STATE_IN_PROGRESS){
            if(associations[i].lifetime < lifetime){
                lifetime = associations[i].lifetime;
                oldest = i;
            }
        }
    }
```

**Code Snippet 3.4**: *allocate_association()* function - Find the "oldest" entry.

When the loop (Code Snippet 3.3) terminates, there is no free association in the table, but the function knows which one of the existing associations has been created first (the parameter "oldest" in Code Snippet 3.4). Actually, this means that the secret shared key contained in the specific association entry has been established prior to the other keys of the table. Thus, to create a new allocation, we decide to remove the oldest one by assigning the zero value to the MAC address, similar to the lifetime "expiration" and deletion of the entry. The difference is that we do not assign the KEY_STATE_NONE as state value (as in key timeout), but the state is determined by the value KEY_STATE_ALLOCATED and returned as the proper free association for the new key establishment. Moreover, it must be noted that only the oldest entry with state value KEY_STATE_COMPLETE is taken into account for deletion, as shown in Code Snippet 3.3 (a). This decision was taken to protect the allocated entries which are related to a key exchange process in progress.

The flowchart of the *allocate_association()* function is briefly presented in Figure 3.4. This function is a basic component of the key exchange process, as an association of the ACL is necessary to be allocated to be able to establish and store a secret key

.

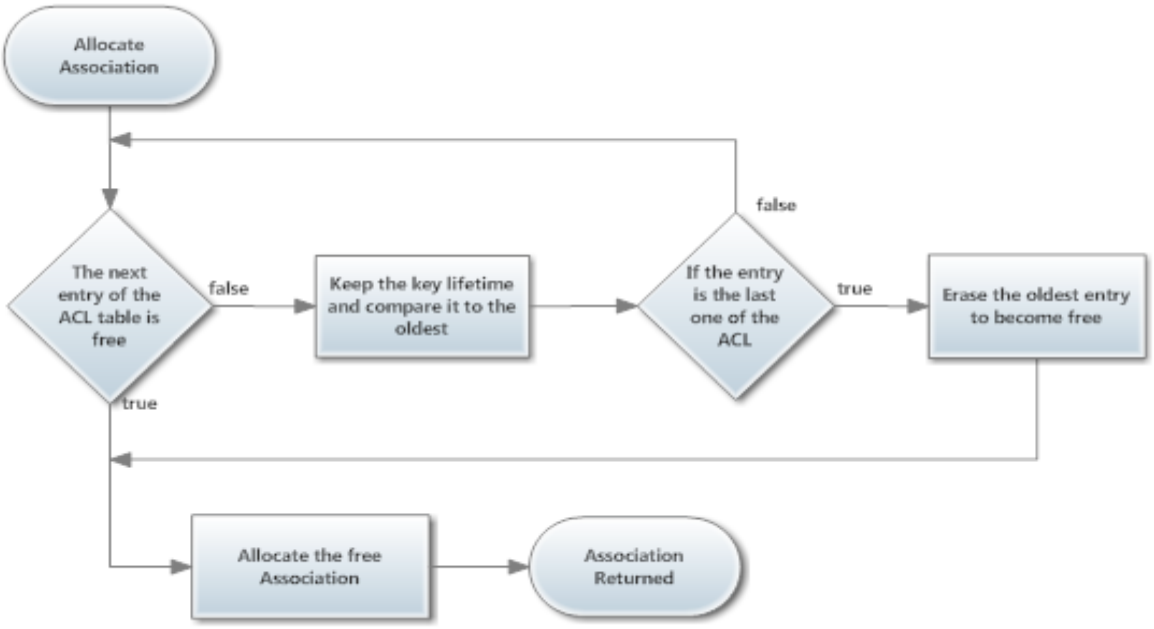


**Figure 3.4**: The flowchart of the *allocate_association()* function.

Furthermore, we construct a solution to overcome the problem of an ACL association being permanently allocated because of indefinitely waiting for a key exchange reply message. We will consider a scenario with two nodes (Alice and Bob) where Alice wants to send a unicast message to Bob. In order for Alice to send Bob a message, she has to establish a secret key with him. First of all, Alice allocates an entry in her ACL table and assigns to the association Bob's local address. After that, as we will see in the next section, she performs the necessary computations to create her private and public ECC keys and sends her public key to Bob. The problem is that the lifetime of the secret key is assigned after the creation of the key and not at the time of the entry allocation. Alice must receive a response from Bob in order to be able to calculate the common secret key. However, it is very possible that Bob goes offline or becomes corrupted and as a result his response is never received by Alice. In such a case the allocated entry of Alice's table does not contain a lifetime value to become "old" enough for deletion or to "expire". The state of the entry will always be assigned as KEY_STATE_IN_PROGRESS resulting to permanent allocation.

To bypass the above problem we have chosen to initialize the lifetime value to a small time interval of 50 seconds by default just after Alice sends her key exchange request to Bob. After the initialization of the temporary lifetime (for the process of key exchange), the lifetime decreases properly every clock second by the *purge_association()* function, similar to how the secret key lifetime decreases. If the key exchange lifetime reaches the value of zero and the process is not complete, it means that a problem occurred on Bob and thus the allocated entry on Alice's table is released.

The associations three (3) and four (4) of the example ACL table meet the above situation. Actually, by observing the third entry of the Table 3.1 we examine that Alice allocated the entry by assigning Bob's MAC address. After that, she created her public ECC key and sent it over the network to Bob. When the key exchange request is sent, the state of the corresponding entry becomes IN_PROGRESS and the temporary key exchange lifetime is set as the lifetime of the association. As shown in the table, Alice waits for the key exchange reply from Bob for 8 seconds, as the lifetime of the entry has the value of 42 (initially the value is set to 50 seconds).

The forth (4) entry of Table 3.1 is similar to the previous one, but with the only difference that Alice has received the key exchange reply from Bob and is currently computing the shared secret key. This is determined from the fact that the Public ECC Key of the specific ACL entry is not NULL, containing the public key of the opponent sensor node. By observing the entry, we conclude that Alice has sent the key exchange request 17 seconds earlier (the temporary lifetime has the value 33), that the reply message from Bob is received and that Alice currently is computing the secret key, while the state is still IN_PROGRESS.

In the proper case where the key exchange process finishes before the temporary lifetime expiration and the secret key has been established, the lifetime of the association is updated to the default key lifetime. The above functionality is necessary for Alice, as she sends the request and waits to receive a response. In contrast, Bob does not need to follow this functionality as he performs the operations in a sequence and does not wait for any message to be received.

The device's state transition diagram for the process of key exchange is presented in figure 3.4. As shown, the device is mainly listening of the medium. When a frame is to be sent, the framer calls the get_key_for_neigh() function and moves to the "Searching for the secret key" state. If the proper entry exists in the ACL table and the key is found, a pointer to the key is returned and the device goes to listening state again. If the key does not exist, we return NULL and we move to the state "Starting Key Exchange". Moreover, as it can be observed (in the top-right corner of Figure 3.5) every second the device enters the state "Purging Expired Associations", where the lifetime of the associations is updated. After the lifetime update the device returns to the "Listening" state. The

"Purging Expired Association" state means that the device does not only update the lifetimes of each entry, but also erases entries whose lifetime reach the value of zero.

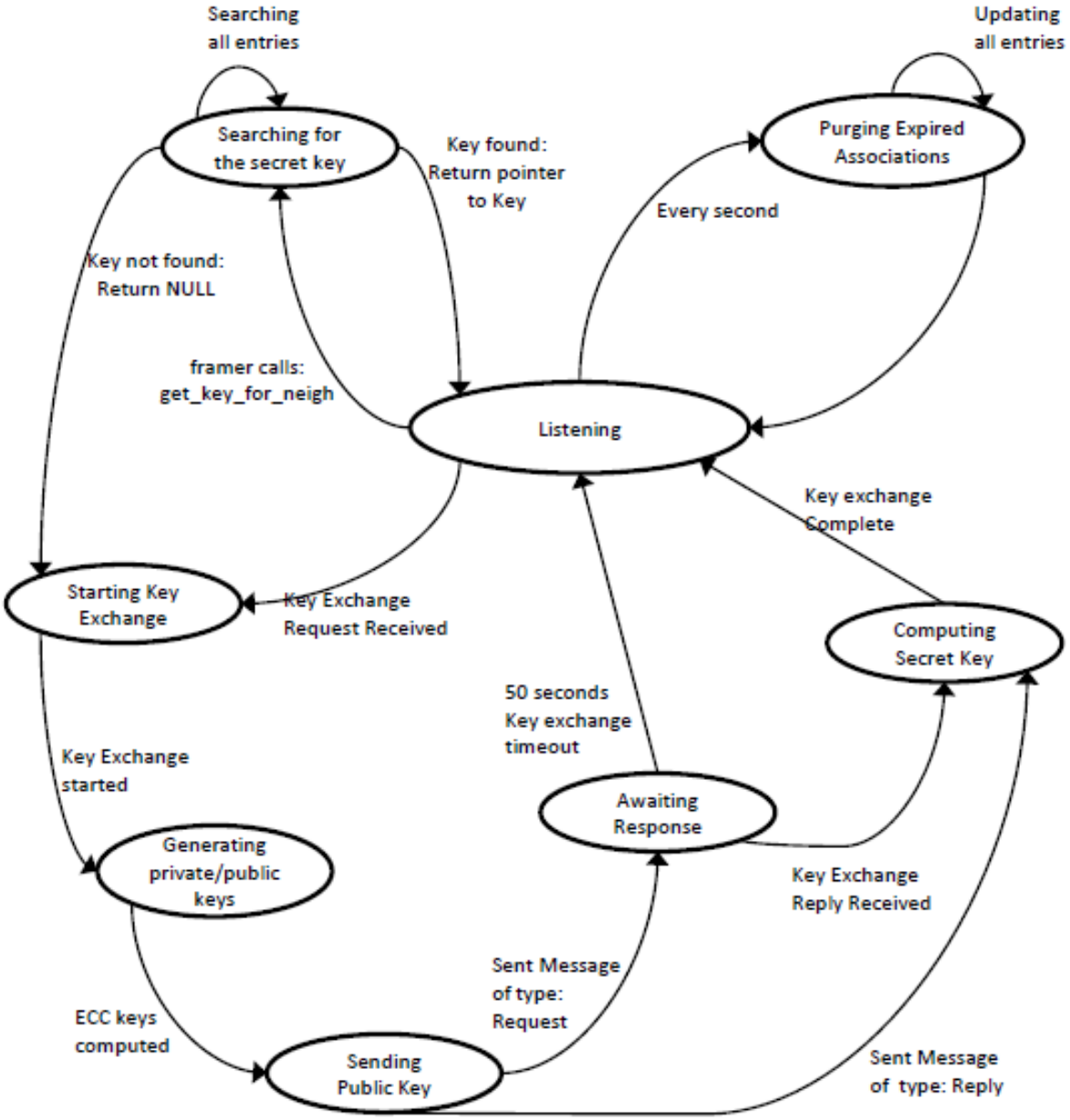


Figure 3.5: The state transition diagram of the device for the key exchange process.

Also, as mentioned above, we implemented a mechanism for assigning a lifetime value to the key exchange process in order to avoid the situation of permanently allocated entries. The ACL entries which are IN_PROGRESS (assigned the key exchange lifetime) are handled in the "Purging Expired Associations" similar to the COMPLETE entries. The key exchange lifetime is updated in the same way as the secret key lifetime and the entries are erased when the lifetime is expired.

By observing the state transition diagram of Figure 3.5, it can be seen that the device remains in the "Awaiting Response" state only for 50 seconds, waiting for the key exchange reply message and then

moves to the "Listening" state. This happens as the lifetime of the key exchange expires and the corresponding association is released.

### 3.2.4 The key exchange (ECDH) process

The Diffie-Hellman key exchange process is a transparent process, actually a daemon, which always runs at the background of the sensor devices. The process is started at the beginning of the Contiki network example (the application) and remains in silence as long as a key exchange is not requested. It is of great importance that the process is independent from the application and more specifically, that the application does not even know about the existence of the key exchange process. Moreover, the ECDH process uses a UDP connection with a default unique port, which is different from the ports used by the application in order to avoid confusion and communication problems.

As described earlier, the framer of the MAC layer asks to receive the secret encryption key by calling the *get_key_for_neigh(mac_address)* function. This function, as presented in Code Snippet 3.2, searches the ACL table to find the corresponding entry that contains the specific MAC address and returns a pointer to the key. If the proper entry does not exist, it returns the NULL value and asks the ECDH process to establish a new secret key by calling the *start_key_exchange* function of the sender node (Alice). The ECDH daemon is already having an established UDP/IP connection that uses the default UDP port for key exchange, which is initialised at the network creation time.

The *start_key_exchange(mac_address)* function has the duty to trigger the ECDH daemon in order to start the key exchange procedure. When the specific function is called, Alice tries to allocate an entry on her ACL table and if the allocation is successful and an association is returned, she assigns Bob's MAC address to the allocated association. However, the IP address of Bob is needed for UDP/IP communication in order to achieve secret key establishment, so Alice uses its Neighbor Discovery table (ND) to lookup Bob's IP address. It must be noted that the ND is an already existing component of the Contiki OS and is mainly used for network construction and routing purposes.

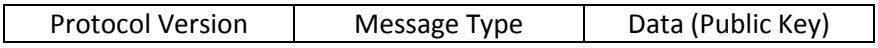


**Figure 3.6**: The format of the messages that are sent during the key exchange process.

If the returned IP address is valid and the ND state of the neighbor node is not incomplete, Alice proceeds to compute her private ECC key as well as her public key and enters the "Generating the private/public keys" state, as shown in Figure 3.5. After the computation of her public key, her public key is loaded into the buffer in order to be sent to Bob, as shown in Figure 3.6. Also the protocol version is loaded into the buffer to ensure protocol compatibility along with the message type value, which is determined as MSG_TYPE_REQUEST. After the creation of the key exchange request message, the contents of the buffer are sent to Bob by using the ECDH UDP/IP connection. When the transmission is completed, Alice defines the key-state as KEY_STATE_IN_PROGRESS, sets the lifetime value as the temporary key exchange lifetime value (50 seconds) and waits for Bob's response. When her request message is transmitted, she immediately enters the "Awaiting Response" state.

The simple protocol we have built for the format of the messages, which are sent during the key exchange process is shown in Figure 3.6. The first field of the message contains the version of the used Protocol (1byte) in order to ensure the compatibility of the communicating nodes. The second field contains the type of the message that is sent, which takes one byte in memory. If the node sending the message is the one that has started the key exchange process, the message type value is

set to MSG_TYPE_REQUEST. If the specific message is sent as a response to a previous request message received, its type is defined as MSG_TYPE_REPLY. By that, we know if the received message corresponds to an already running key exchange process or to a new one. The size of the payload field (Public key) depends on the used elliptic curve, as the two coordinates of a specific point on the curve are sent. In the current implementation, the size of the payload is 32 bytes as the used curve is the SECP128R1 of 128 bits.

On the other hand, the key exchange process does not only start when the function *start_key_exchange(mac_address)* is called, but also if a key exchange message from a neighbor node is received by the ECDH daemon. The daemon is always listening to a specific port in order to receive any key exchange messages having the sensor's address as the assigned destination address. These messages are represented as TCP/IP events in the Contiki OS.

On the reception of an ECDH message, the receiver node checks if the type of the used protocol version is supported and if the type of the received message is either MSG_TYPE_REQUEST or MSG_TYPE_REPLY. After that, by knowing the sender's IP address it tries to match that with an entry in its ND table. If the neighbor is known (exist in the ND table) and its state is complete, the message is accepted for further process.

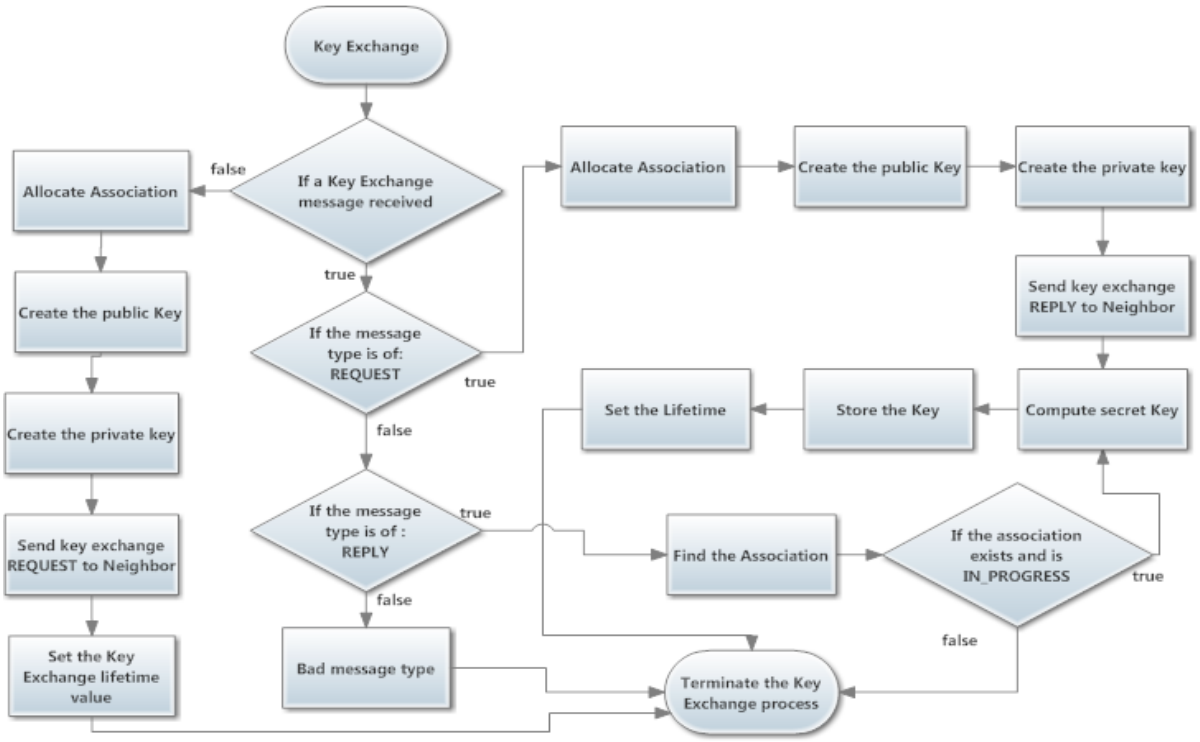


**Figure 3.7**: The flowchart of the Key Exchange process.

At this point, if the type of the received message is MSG_TYPE_REQUEST, it means that the sender node is asking for key exchange and sends its public key. The receiver (Bob) tries to find and allocate an empty entry in his ACL table to create an association and copies the sender's MAC address and public key to that entry. After that, Bob computes his private and public ECC keys and as Alice previously did, he loads his public key into the buffer. Moreover, he sets the version of the used protocol, sets the message type value to MSG_TYPE_REPLY and sends the message, as shown in Figure 3.7.

After the message is sent, he computes the secret shared key by multiplying the received public key with his private ECC key. Subsequently, Bob sets the key state of the specific ACL association to KEY_STATE_COMPLETE and the key lifetime to the default lifetime value. Then he erases the public key of Alice as it is not needed any more and his private key from the association. Figure 3.8 presents the individual steps of the key exchange process and the sequence of the transmitted messages, both for Alice and Bob.

In the case where the received message has the value MSG_TYPE_REPLY in the message type field, it means that the message was sent by the receiver node (Bob) back to the node which started the key exchange process (Alice). This message was sent as a response to a previous sent request. In such a case, the node receiving the message calls the *find_association (neighbor_mac_address)* function in order to match the sender's MAC address to an existing MAC address in the ACL table, as shown in Figure 3.7. If there is a previously created association, containing a matching address and the state of the key is defined as KEY_IN_PROGRESS, the receiver knows that the current message is a valid reply to its previous key exchange request. Thus, it computes the secret key by multiplying the received public ECC key with its private ECC key found in the association entry of the table. After the computation of the secret key is complete, its private key is erased, the key state is marked as KEY_STATE_COMPLETE and the key lifetime is updated to the default value.

It is also possible for a sensor node to receive a massage of type MSG_TYPE_REPLY, but the sender's MAC address not to match the address of any allocated association. Consequently, the message is ignored due to the fact that the reply message was received a long time after the transmission of the request message and after the key-exchange lifetime value has expired. This problem can be caused by congestion of the network, by the large number of key exchange processes a node has to handle at the same time or by multi-hop communication and routing problems.

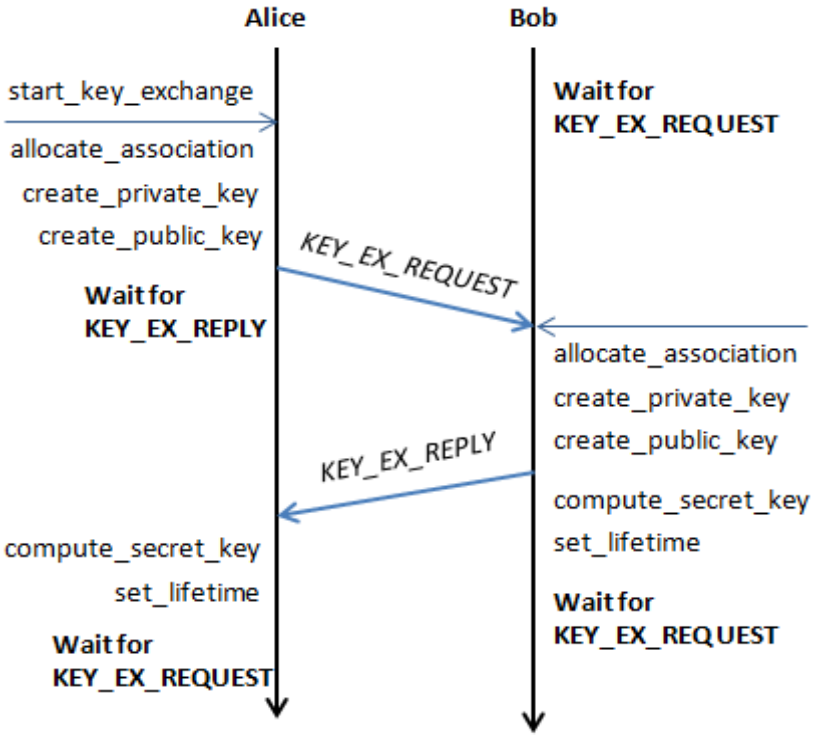


**Figure 3.8**: The key exchange process - full "Handshake"

As mentioned earlier, the ECDH daemon is always running in the background waiting for events. We have developed the ECDH daemon as a separate thread that starts running at the start-up of the sensor devices and establishes a UDP connection, as shown in Code Snippet 3.5. Moreover, the process remains in silence, listening to the wireless medium for key exchange messages.

```
PROCESS_THREAD(ecdh_key_process, ev, data) {

    PROCESS_BEGIN();

    memset(associations, 0, sizeof(associations));
    ecdh_conn = udp_new(NULL, UIP_HTONS(KEY_EX_UDP_PORT), NULL);
    if(ecdh_conn == NULL) {
        PROCESS_EXIT();
    }
    udp_bind(ecdh_conn, UIP_HTONS(KEY_EX_UDP_PORT));

    ecc_init();

    etimer_set(&lt, CLOCK_SECOND);
    while (1) {
        PROCESS_YIELD();
        if (ev == PROCESS_EVENT_TIMER && data == &lt) {
            purge_associations();
            etimer_set(&lt, CLOCK_SECOND);

        } else if(ev == tcpip_event) {
            tcpip_handler();
        }

    }
    PROCESS_END();
}
```

**Code Snippet 3.5**: The key exchange process thread (The ECDH Daemon).

As it can be seen, when the ECDH daemon starts, it initializes all the associations of its ACL and establishes the *ecdh_conn,* which is a UDP connection on a specific port (KEY_EX_UDP_PORT). After that, it initializes the elliptic curves in order to construct the used curve and get its base point. Moreover, it sets a timer with the value of one second and enters the main loop, waiting for events. The key exchange daemon is able to handle events of two different types: TCP-IP events, where a message is received and time events.

In the first case, when a time event occurs the daemon calls the purge_association() function, which updates the lifetime of the ACL associations and after that, it resets the timer. By doing this, the daemon keeps its ACL updated, containing only the valid associations. In the second case, when a TCP-IP event occurs, which means that a message is received, the daemon calls the tcpip_handler() function to handle the received message.

The tcpip_handler() function initially checks if the protocol version is supported and if the message is of type REQUEST or REPLY. If the message is of type REQUEST, it gets the data of the message, computes and sends the public key and then later, computes and stores the secret key. If the message is of type REPLY, then it only computes and stores the secret key, otherwise the message is discarded. Actually, the tcpip_handler() is the most basic function, as it performs the key exchange procedure, as shown in Figure 3.7.

## 3.3 Discussion

It is common practice for each sensor node to create its ECC keys only once, during the initialization phase of the network and store these for future use. These two keys, which are used for the secret key computation, are stored permanently and they cannot be changed. If an adversary or a malicious node is eavesdropping on the network and achieves to gain knowledge of two different public keys, it is not able to determine their secret shared key. The adversary needs to know at least one of the two private keys to multiply it with the public key of the other node in order to compute the secret encryption key. Despite that, it takes exponential time for the adversary to "break" a point on the elliptic curve and extract the node's private key, knowing the base point of the curve and the public key.

In contrast to this common practice, we have decided not to store the ECC private and public keys of the sensors internally, but to discard them after the computation of the secret key. Thus, for every new key establishment, the participating sensor nodes must compute their new private and public keys. The main drawback of the solution we have implemented is that the sensors consume a larger amount of energy and time to compute the secret key, in contrast to the aforementioned common practise. This happens because every sensor node needs to perform two EC multiplications, one for the public key computation and another for the secret key, instead of only one in the case where the public key is stored and remains unchanged. Another drawback is that the first sensor node, asking for key exchange, has to wait until receiving the response of the second node to compute the secret encryption key. If the ECC keys were permanently stored, we could avoid having synchronisation and time spent on waiting.

However, the benefits the current implementation provides outweigh the energy and time consumption drawbacks. First of all, the private key of the sensor device cannot be leaked as it is not stored permanently, but in the scenario where it is leaked, it will not be useful for a long time as the secret keys have a limited lifetime. Moreover, every new secret key is computed according to a new and different private key, so the leakage of the private key can affect the security of a single secret key. Furthermore, by always having different and changing ECC keys, we are able to implement a key timeout mechanism to avoid giving the adversary the advantage to perform cryptanalysis methods.

Apart from the strong security guarantees, the use of different ECC keys provides network scalability and limited memory consumption as the key associations of the ACL table are erased when the timeout of the secret key occurs. Otherwise, we would have used an association table of larger size and built a stronger and more complicated replacement mechanism, not only finding the empty or first created entries, but also replacing the entries that have been used the least.

## 3.4 The application layer example

The application we used at the application layer of the network stack was an already existing example of the Contiki OS, which is slightly modified in order to start the ECDH daemon on the start-up of the devices. The used example consists of two different parts, specifically the Contiki example files *udp-server.c* and *udp-client.c*. Both the server and the client are running a process, where the former one provides the server side functionalities and the latter processes the operations of the client. Therefore, we can build a complex network example with one sensor node running the server-side application and multiple sensor nodes acting as clients of the network.

All the nodes create a UDP/IP connection when their process thread begins at the initialization phase of the network. This connection remains active as long as the process is running. The server connection is configured to always listen to the port 5678 (SERVER_PORT) and to send its reply messages to the client port 8765 (CLIENT_PORT). On the other hand, the client is configured so that

its UDP connection listens to the CLIENT_PORT and the outgoing messages are sent to the SERVER_PORT.

Moreover, the Neighbor Discovery operation is performed at the initialization phase of the network. It is a distributed operation, performed by each specific node in order to discover its neighbouring nodes and construct its routing table. As mentioned in section 2.1.3.1, the server node has the responsibility to advertise the IPv6 address prefix, which is given to every internal router and in turn, every router distributes the prefix to the nodes. The server node is set as the root of the rooting table, but as the functionality of the RPL protocol is distributed every node constructs its own routing table. If a routing problem appears during the network running time, a specific event occurs at the server node that is responsible to take the necessary actions to repair the network, by calling the rpl_repair_root() function, as shown in Code Snippet 3.7. This function is part of the routing protocol and thus it is pre-existing in the Contiki OS.

The core functionality of the server process is simply to detect the UDP/IP events and handle properly the incoming messages. The server process receives multiple unicast messages, sent by multiple clients at the same time, and counts them. We did not enable the server node to send reply messages back to the clients or to take any other actions in order to keep the application example simple. Actually, the server node does not start the key exchange process as it does not send any messages to the other node, but only operates when a key exchange request is received.

On the other hand, the nodes running the client-side application are configured to have the IP address of the server as the default address of the UDP/IP connection. The main role of the clients is to periodically send messages to the server. The transmission periodic interval is set to 30 seconds, so that the client nodes are constructing and sending a message every 30 seconds. The outgoing messages contain the ID of the sender node as well as the sequence number of the message.

Moreover, it is important that the client application is responsible for defining whether the outgoing messages should be encrypted or not. Therefore, after a message is created and loaded into the buffer and before it is transferred to the lower layers of the network stack, the *frame_enc* value is set to request for encryption. When the message is on its way to transmission, it eventually reaches the MAC layer and specifically the framer. The MAC framer checks whether the outgoing frame should be encrypted or not and if it is, it asks for the secret encryption key, as previously discussed in section 3.1.

```
etimer_set(&periodic, SEND_INTERVAL);

while(1) {
    PROCESS_YIELD();

    if(etimer_expired(&periodic)) {
        etimer_reset(&periodic);
        ctimer_set(&backoff_timer, SEND_TIME, send_packet, NULL);
    }
}
```

**Code Snippet 3.6**: The client node process thread, which sends application messages

In the Code Snippet 3.6, we can observe the main functionality of the client process thread. Except from the initialization of the process and the establishment of the UDP connection, the client sets a timer with a periodic send interval at 30 seconds. After that, as it can be seen, it enters the main

loop where it waits for events to occur. The events the client is able to handle are the periodic time events of the timer. Actually, every time that an event occurs, the process of the client node sends a message to the server and then it resets the timer in order to generate the next time event.

```
while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
        tcpip_handler();
    } else if (ev == sensors_event && data == &button_sensor) {
        printf("Initialising global repair\n");
        rpl_repair_root(RPL_DEFAULT_INSTANCE);
    }
}
```

**Code Snippet 3.7**: The server node process thread, which receives application messages

On the other hand, the main functionality of the server node, as mentioned earlier and shown in Code Snippet 3.7, is to receive the incoming application messages of the clients. Similar to the ECDH daemon, this process enters the main loop and always listens for events. The main events it receives are TCP-IP events, where the handler function is called. Actually, the tcpip_handler() is only receiving messages, prints them and increases its counter, which keeps the number of the received messages. Moreover, the server handles events caused by the network that report routing problems, and starts the rpl_repair_root() function, which performs a global repair operation on the entire network.

# 4. EXPERIMENTAL SETUP, RESULTS AND ANALYSIS

In this chapter, we present a number of different experiments carried out in order to provide the results and the statistics for our implementation. Additionally, these results will be the basis of our analysis and discussion as well as to conduct a critical evaluation of the implemented solution.

The first and simplest experiment we have designed consists of only two nodes communicating for a long time period. During the communication, the sensor nodes re-establish a new secret key every time (more than ten re-establishments) the lifetime of the previous key has turned to zero. The scope of this experiment is to address the energy consumption and the time required for the computation of their public and secret shared keys as well as for a full key exchange process to be completed.

In the second experiment, we build a network of ten (10) nodes, where one of them acts as a server and the rest act as clients. In this experiment we check how the value of the key lifetime affects the performance of the network by calculating the packet-loss ratio. The packets are lost due to the absence of a secret key or because the key exchange process is in progress. Additionally, we allow the simulation to run multiple times, for exactly the same time, and we increase the key lifetime period every time a new simulation is run.

Moreover, as we gradually increase the key lifetime, we address the energy consumption of both the devices that act either as a server or as a client. We estimate the overall energy consumption as well as the energy spent by each separate module of the sensor nodes. Also, we examine the difference on the energy consumption if the node acts as a client or server as well as if it performs message routing operations.

The next experiment aims to test the scalability of the network in relation to the key exchange process and the number of the established keys. This experiment is carried out using the network of the previous scenario with the difference that the established keys have fixed key lifetime value for all the sensor nodes. Moreover, we gradually change the size of the ACL table of the sensor nodes and measure the memory consumption as the size increases.

The parameters used in the simulations are given in Table 4.1. All the given parameters remain unchanged in every simulation if it is not clearly stated on the description of the specific experiment that the configuration has been changed.

| Parameter | Value |
|---|---|
| **Motes** | Tmote Sky |
| **Range** | TX: 50m |
| **MAC Layer** | IEEE 802.15.4 |
| **Radio Access** | CSMA |
| **Duty Cycling** | ContikiMAC |
| **Queue buffer size** | 4 |
| **Max neighbors** | 4 |
| **ACL size** | 4 |
| **ACL entry lifetime** | 1000 seconds |
| **key exchange lifetime** | 50 seconds |

**Table 4.1**: Simulation Configuration

## 4.1 Memory requirements

The available memory of the sensor devices is very limited, as mentioned earlier. Specifically, we use the Tmote Sky devices for the simulation experiments, which provide only 48 KB of flash memory (ROM) for source code hosting as well as 10 KB of RAM. Furthermore, the available memory of each device is less than the provided memory as a large amount of memory is used to host the source code of the Contiki OS. The memory consumed by the Contiki OS can be reduced by disabling the unused or unnecessary functionalities, but cannot be eliminated at all. This happens because a large portion of the source code of the Contiki OS is required for communication over the Internet protocol or to reduce the consumption of energy by implementing the duty cycle mechanism of the ContikiMAC.

To be able to implement a key exchange mechanism with the ECDH technique we had to ensure that the 16-bit microcontroller of the device is capable of supporting numerical operations with large numbers, which are represented as multiple words. As mentioned earlier, we used the *nn.c* file, which is provided by the ContikiECC project, for the implementation of the basic and necessary numerical operations. Moreover, we used the SECG - standardized elliptic curve SECP128R1, which is implemented in the secp128r1.c file of the ContikiECC, as well as the file ecc.c. The file secp128r1.c provides the proposed elliptic curve by defining the parameters *a* and *b* of the curve as well as specifying its base point. The ecc.c file provides functions for a number of operations over the elliptic curve, performing point additions, point doublings and scalar multiplication. However, the source code given in the ecc.c file requires a large amount of memory, as it provides many different functions and operations mainly for computation optimisations. In order to reduce its size, we only use the absolutely necessary functions.

Moreover, we have modified the structure of the MAC layer frame of the ContikiOS to support the IEEE 802.15.4 auxiliary security header. Not only that, we also modified the MAC framer to ask for the encryption keys in order to secure the outgoing frames. These small modifications provide the MAC layer with important functionality, but they do not increase the size of the source code dramatically. In comparison to the pre-existing version of the frame and the framer, the increase of the source code takes inconsiderable size.

The source code of the ECDH file is responsible for performing the key exchange operation between the sensor nodes by implementing an always running process, ready to establish a new secret key when needed. Also, this file implements the ACL table of the nodes, containing multiple entries, each having a pair of a MAC address and a secret key. Moreover, it provides functions for searching the ACL entries in order to return the key, to allocate, delete or replace an entry of the table as well as to keep and maintain the lifetime of the keys.

| File name | ROM (text) | RAM | | Overall |
|---|---|---|---|---|
| | | data | bss | |
| **nn** | 3372 | 0 | 0 | 3372 |
| **ecc** | 2494 | 0 | 676 | 3170 |
| **ecdh** | 1477 | 10 | 392 | 1879 |
| **secp128r1** | 402 | 0 | 0 | 402 |
| **Total** | **7745** | **10** | **1068** | **8823** |

**Table 4.2**: Memory requirements of the used files, in bytes

The memory requirements of the above files are given in Table 4.2, which provides both the ROM memory needed for source code hosting and the RAM memory consumed. As it can be seen, the ROM memory that is spent for handling the large numbers and performing the basic numerical operations is about 3.3 kb. Moreover, we spend about 4.3 kb of ROM, as well as 1kb of RAM to implement the operations over the elliptic curves and develop the key exchange process.

The memory consumed for the implementation of the key exchange process is acceptable, as only 7.7 kb of ROM memory is required for the source code. Probably we could be able to use the SHA-1 algorithm along with ECDSA, having additional memory consumption only at 3-4 kb for hosting of the source code. If we manage to use ECDSA in the future, we would ensure mutual authentication of the nodes during the key exchange process.

Moreover, a software implementation of the AES algorithm takes 3.9kb of ROM memory, so we do not currently use it. However, we have plans to implement it on the hardware, as the specific sensor devices provide a co-processor capable to perform the AES encryption. By implementing the AES in the hardware, we avoid the memory consumption the software implementation requires.

## 4.2 Experiment: Key exchange with ECDH

The current experiment is the simplest one of the experiments we have performed, but of great importance as it provides measurements about the core elliptic curve techniques we have used. It assesses the performance and provides statistics for a full key exchange between two nodes. The experiment has been designed having only two sensor nodes (Alice and Bob) communicating for a long period of time. At the first, Alice wishes to send a message to Bob, but she finds that there is not an entry in her ACL table matching Bob's MAC address. Consequently, the message is dropped by the framer of her MAC layer and the key exchange process, which is responsible for key establishment between the two nodes, starts. In this example, Alice is always the sensor node that begins the key exchange process (represented as the node 1 in Figure 4.1) and Bob is always the node receiving Alice's request and replies with his private key (the node 2 in the Figure 4.1).

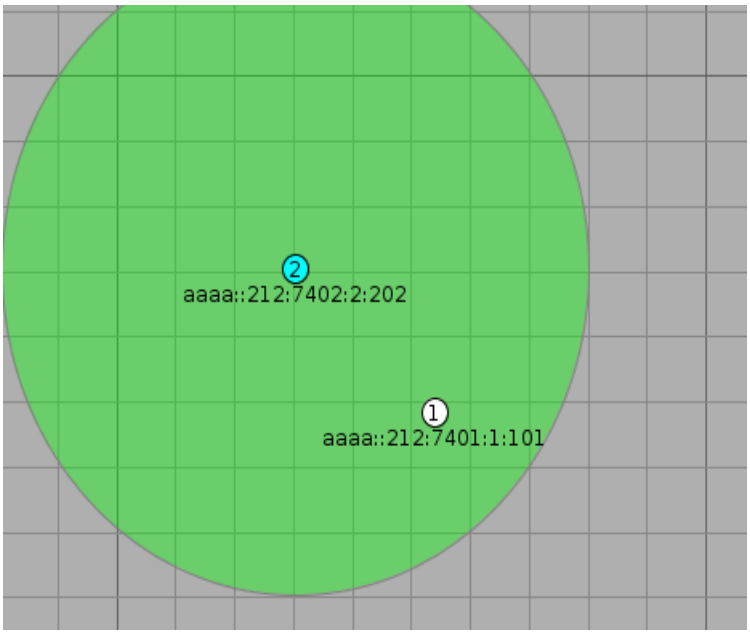


**Figure 4.1**: The experimental setup of the key exchange process (2 sensor nodes).

When the two devices share a secret key, the communication is performed properly as long as the lifetime of the key has not reached the value of zero. When the secret key expires, the entry of the ACL table of the nodes is erased and thus a new key exchange process must start when a new message is proceeded to be sent. Specifically, the sensor nodes communicate over a long period of time, in order to face multiple key timeouts and multiple key establishments. The overall time the experiment is run, is enough to count more than 15 key establishments between the two nodes.

As discussed earlier, we decided not to permanently store the public key of the sensor nodes, but to discard it and create a new one when a secret key is requested. So, the nodes spend time and energy not only to compute the secret key but also to create their new public key when the key exchange process begins. Both calculations are performed by a scalar multiplication of a point on the elliptic curve with a random 128-bits number (the private key of the node).

With this experiment, we estimate the amount of energy that is spent for the creation of the public key and the computation of the shared secret key. Specifically, the energy and time spent for these two computations, correspond to the energy and time needed for the elliptic curve operation of point scalar multiplication. Both experiments were run for enough time to perform more than ten key establishments, so that we can describe the given time and estimate the energy consumption by their average values.

The time (in seconds) needed for the creation of the public key and the computation of the secret key is given in Table 4.3, as well as the overall time the process of key exchange takes. The specific table provides the time both for Alice, which starts the process of key establishment, and for Bob, which only responds after a key exchange request is received.

| Time (s) | Public key | Secret key | Key Exchange Process |
|---|---|---|---|
| **Alice** | 8.560 | 8.547 | 25.586 |
| **Bob** | 8.457 | 8.416 | 16.873 |

**Table 4.3**: Average Time consumption of the key exchange process

| Energy (mJ) | Public key | Secret key | Key Exchange Process | |
|---|---|---|---|---|
| | | | MCU | MCU + TX + RX |
| **Alice** | 47.176 | 47.226 | 97.021 | 113.224 |
| **Bob** | 47.165 | 47.171 | 94.353 | 100.758 |

**Table 4.4**: Average Energy consumption of the key exchange process

The Table 4.4 presents the energy Alice and Bob consume to create their public key as well as the shared secret key, similar to the previous table. Moreover, it presents the overall energy consumption for the full key exchange process. The energy spent for the creation of the public and the secret key is estimated by the average energy consumption of the device's microcontroller unit. Actually, we do not want to estimate the overall energy consumption of the device (microcontroller, transmitter and receiver) as the specific part of the experiment focuses on the assessment of the elliptic curve point multiplication operation.

Furthermore, the above table provides the amount of energy that is consumed for a complete key exchange. The presented results address two different situations, where the first one counts only the energy consumed by the microcontroller unit and the second estimates the overall amount of

energy that is consumed by the different modules of the device. We have decided to estimate the energy consumption for both cases because of the fact that the transmitter and the receiver of the devices are needed to achieve communication during the key exchange process.

Moreover, it is important to assume that the components of the device such as the micro-processor, the transmitter and the receiver work on average medium workload. The energy estimation is achieved by taking measures of the working time each single component requires to perform a specific operation. After that, we calculate the energy consumption of each component in accordance to its average workload, as it is specified in the Tmote Sky specification datasheet [66]. However, if we know that an operation is very demanding and that the average workload of the micro-processor would be high, we can estimate the amount of energy by applying a larger consumption factor. The opposite can be applied (use of small average consumption factor) if we know that the specific operation is not MCU demanding.

### 4.2.1 Experimental Analysis

It is observed that the computation of both keys and consequently, the operation of elliptic curve multiplication take about 8.5 seconds on average. Having in mind that the used elliptic curve and the multiplier have size of 128-bits, the required computation time is acceptable. Also, the presented time includes the time that is spent for pre-computing all the values of the sliding window, as the implemented elliptic curves use the sliding window method to perform the multiplications. Comparing the time that is spent in the current implementation to the times presented in the literature, we observe that there is potential for improvements by implementing some targeted optimizations.

Moreover, it is observed that the time spent for the computation of the secret key is slightly less than the time spent for the creation of the public key, for both Alice and Bob. This happens because in the case of the public key computation, it is necessary to load the base point of the used elliptic curve from the corresponding file containing the parameters of the curve. In contrast, in the computation of the secret key, we use the public point of the opponent sensor node, which is received and loaded directly from the key exchange communication.

Furthermore, Alice consumes slightly more time on average than Bob (about 0.117 seconds) for the computation of her ECC public key and secret key. The small difference on the time consumed by the multiplication operation of Bob and Alice is mainly due to external factors. For example, Alice acts as a client node, asking for the encryption keys and trying to send a message to Bob every 10 seconds. The attempts for message transmission by the application, take place at the same time the key exchange process is in progress. These functionalities, as well as the routing processes, are performed by different and distinct running threads. As mentioned earlier, the ECDH process is autonomous, runs in the background and is handled by a specific thread, where the application process (performs message transmission) is handled by a different thread. Eventually, all the background processes and running threads consume energy and time of the single device's microcontroller, which actually shares its time between the served threads. This is the reason that the time Bob consumes is slightly less than Alice's, because Bob does not have a process running as a different thread, sending messages from the application layer with small sending time intervals.

It is also very important to examine the overall time the key exchange process takes for both Alice and Bob. As mentioned earlier, the key exchange procedure starts by Alice when she wishes to send a message to Bob. This means, that Alice computes her private and public ECC keys, sends the public key to Bob and waits for his response. On the other hand, after the key exchange request (public key of Alice) is received by Bob, he computes his private and public keys, sends his public key as a response to Alice and calculates the shared secret key. As observed in Table 4.3, Bob finishes the key

exchange process and generates the secret key 16.87 seconds after receiving the key exchange request. The overall time Bob spends can be roughly estimated as the time that is needed to perform two point multiplication operations.

In contrast to Bob, Alice spends significantly more time than Bob to finish the key exchange process and gain knowledge of the secret key. Roughly, the consumed time corresponds to the amount of time that is needed for three point multiplication operations to be computed. This happens as Alice sends the key exchange request and waits for Bob's response. As stated, we have decided not to keep the ECC keys (private and public key) permanently stored, but to discard them and compute a new pair when needed. Consequently, Bob does not have a public key when a request for key exchange occurs, so it starts computing it. Meanwhile, Alice waits for Bob's reply, which is actually sent after the computation of the public key. Moreover, if Bob was not only computing his public key before sending the reply message, but also the secret key, the time that Alice waits would be doubled. Thus, for energy and time saving, we do not implement the key exchange process to be synchronous and fully sequential.

Table 4.4 presents the energy consumed for the computations of the public and the secret key. The presented energy stands only for the energy spent by the microcontroller unit and not the energy spent by the receiver or the transmitter, as we only care to assess the operation of the point multiplication over EC. We observe that the public and secret key computations of Alice and Bob consume similar energy, where the small difference between them is insignificant. The reason Alice consumes more energy than Bob is that during the computation of the keys some other operations of the application are taking place (similarly to the time estimations).

Moreover, Table 4.4 gives an estimation of the energy amount that is consumed for a full key exchange process. The given values present the consumption of the microcontroller in the first case as well as the overall consumption of the device for the process in the second case. It is observed that the energy estimation for Bob reflects the energy consumption of the public key and secret key operations. In contrast, the microcontroller of Alice consumes about 3 mJ more energy than that of Bob, because of the waiting time, where other unrelated operations are performed. Not only that, but also the receiver and the transmitter of Alice consume about 13mJ, while she is waiting for Bob's public key. These 13 mJ is mainly because of idle listening of the wireless medium.

## 4.3 Experiment: Lifetime of the secret Key

The current experiment has been designed and built in order to assess how the lifetime of the established secret keys affects the energy consumption of the sensor devices and the performance of the network. The topology of the network we have built, for the current experiment, is given in Figure 4.2. It actually consists of ten nodes, where one of them performs the server-side operations (the node "0") and the rest act as the clients. Specifically, the client nodes have been implemented in order to be able to send a message to the server periodically (every 30 seconds). These messages contain the ID of the sending device and the sequence number of the message. On the other hand, the server sensor node does not perform any particular operations on the application layer, except from receiving the incoming messages and sums them up.

As presented in Figure 4.2, the nodes 3, 4, 5 and 7 are out of the transmission range of the server node and thus the communication between them and the server node is not performed directly. When the network is constructed, at the initialization phase, the specific nodes do not establish a UDP connection with the server, but with one of the internal nodes that is closer. Consequently, in addition to the client-side functionalities the internal nodes (3, 6 and 9) have, they act as routers performing the necessary routing operations for the other nodes. To be more precise, the nodes that perform routing operations do not only serve nodes out of server's range, but also those that

are very close to the server. The routing decisions are taken in accordance to the routing tables, which are constructed at the initialization phase of the network.

However, we are not concerned on how the routing should be performed, as the network uses the Routing Protocol for Low-power devices (RPL), which is already implemented in the ContikiOS. Actually, we do not want to define the entries of the routing table or to affect the decision of the next-hop, but only gain knowledge of the topology to be able to know which nodes are used as internal routers. By having this information, we are able to understand the behaviour of the network and assess its performance.

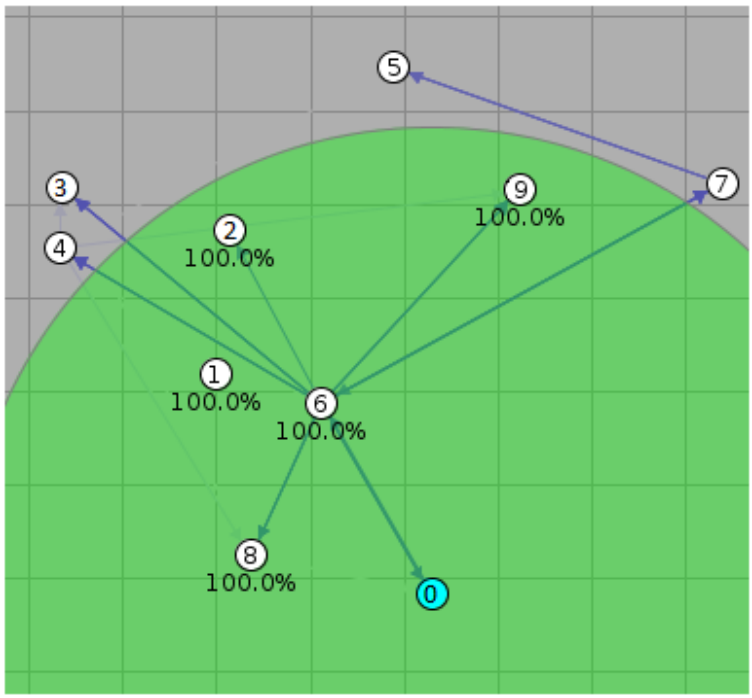


**Figure 4.2**: The experimental setup - Key exchange in a network with 10 nodes

All the simulations were running over a period of two hours, in order to allow a large number of messages to be sent, as well as the key exchange process to be performed many times. In the first simulation, we have defined the key lifetime value as 1000 seconds. This means that the keys are valid only for 1000 seconds after their creation and therefore, when an old key expires a new one must be established to be used for communication. Moreover, it is important to notice that not all the client nodes share a secret key with the server, but only the communicating nodes. Actually, only the nodes which share a connection link, as determined by the routing process, negotiate and establish a shared secret key.

The energy consumption for the first case of the experiment, where the key lifetime is set to 1000 seconds, is given in Figure 4.2 as well as the results of the experiment when the lifetime is set to 1500 seconds. Similarly, Figure 4.3 presents the energy consumption for every sensor in the simulated network for key lifetime values of 2000 and 2500 seconds respectively. The graphs presented in Figures 4.2 and 4.3 show the energy that is consumed by the micro-processor, the transmitter and the receiver of the devices with different colours, but every bar of the chart sums up the overall energy consumption.

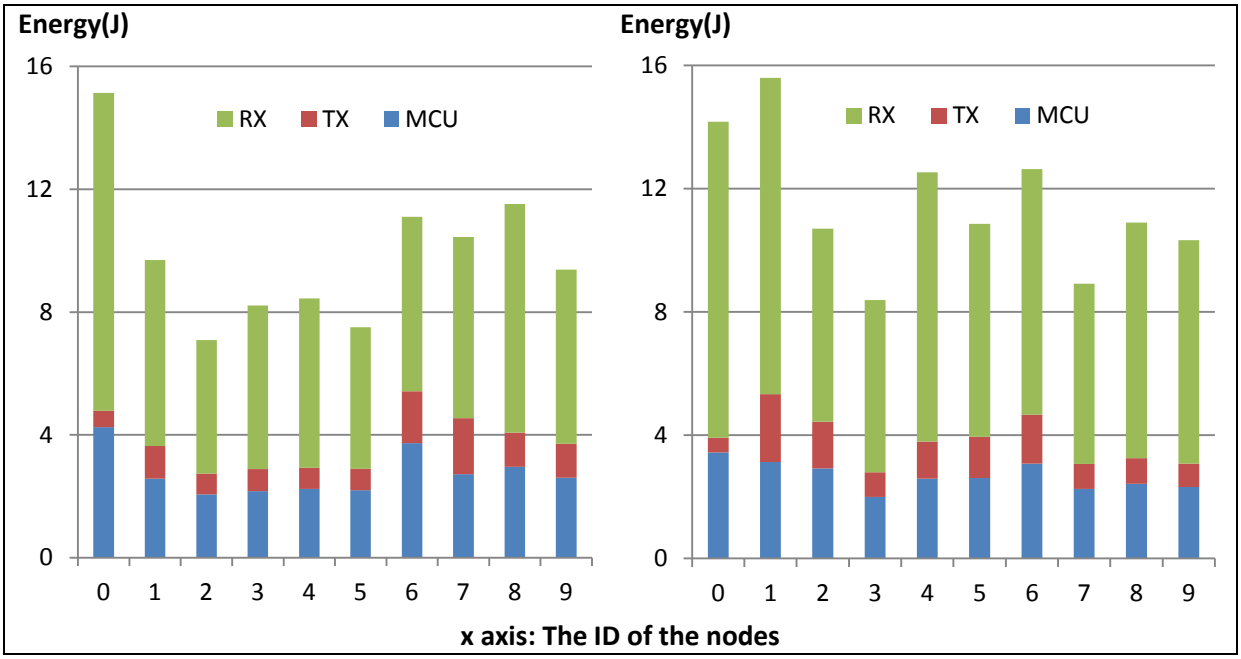


**Figure 4.2**: The energy consumption of the nodes for key lifetime of (a) 1000 and (b) 1500 seconds.

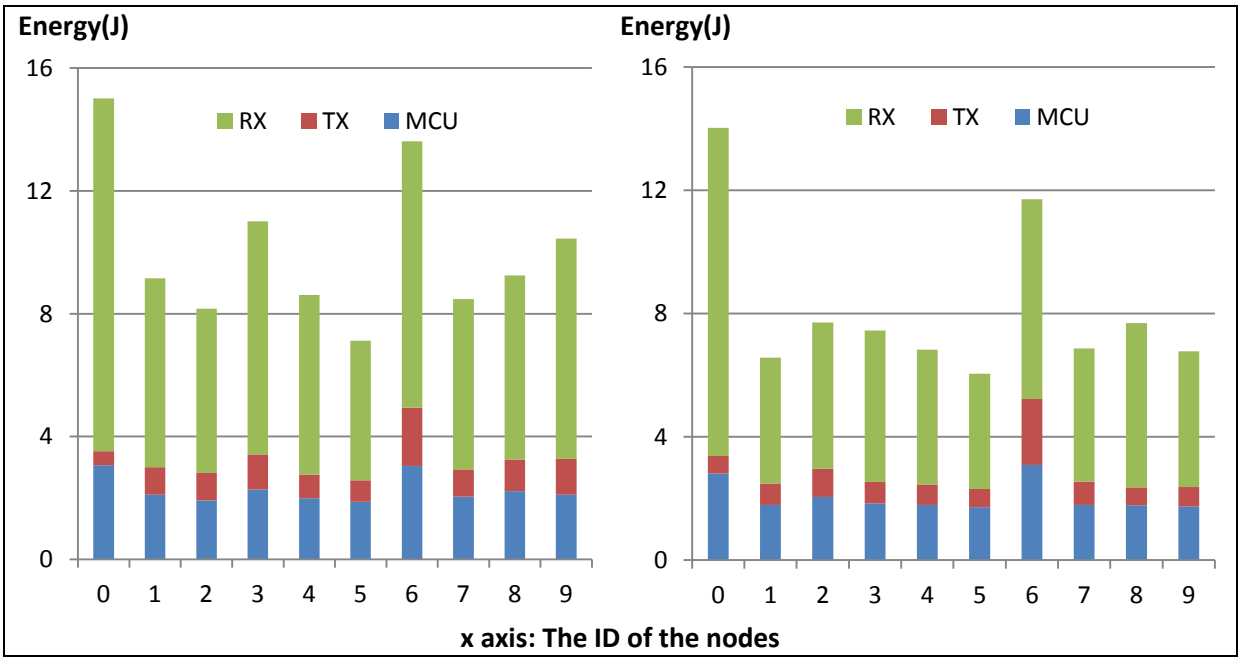


**Figure 4.3**: The energy consumption of the sensors for key lifetime of (a) 2000 and (b) 2500 seconds.

As expected, the energy consumption of the device acting as a server is very high in contrast to the estimated energy consumption of the other devices. Specifically, the server node has the second higher overall energy consumption in one of the experiments and the highest in the other three experiments. If we observe the provided data in more detail, we can examine that the micro-processor of the server node consumes, in each simulation, the largest amount of energy in contrast to the micro-processors of the other sensor nodes. This mainly happens, because the server node

has multiple UDP connections to communicate with the clients and consequently, it establishes multiple secret keys.

Moreover, we observe that the transmitter of the server consumes the smallest amount of energy, in contrast to the other nodes (in all the experiments) and that its receiver spends the largest amount. This is due to the functionalities the nodes implement, as the server transmits only its public key during the key exchange process, while the other nodes transmit their keys and some application messages periodically. Furthermore, the clients do not spent a large amount of energy on radio listening, as the only data they receive are related to the key establishment process. On the other hand, the server always listens to the wireless radio to receive all the incoming messages.

Furthermore, the above figures give information about the nodes which act as internal routers. As it can be seen, the node "6" consumes a larger amount of energy than the other client nodes in all the simulated experiments. Not only its overall energy consumption is larger, but also the energy spent by its micro-processor and its transmitter as well. This happens as the specific node is employed by the network as an internal router. The energy consumed by its micro-processor was spent not only to establish a secret key with the server node, but also to compute the different public and secret keys with the other client nodes it serves. Similarly, the energy its transmitter spends is due to the transmission of the application's messages as well as the forwarding of the messages it receives from the clients.

The routing functionality of the sensor node "6" can easily be examined by an analysis of the energy consumption, as it consumes significantly a larger amount of energy in contrast to the other sensors. This happens because the specific sensor node routes packets from three clients of the network to the server. On the other hand, it is not apparent that also the nodes "3" and "9" are used as internal routers, because these two nodes route packets from a single node and therefore, the energy consumption is only slightly affected.

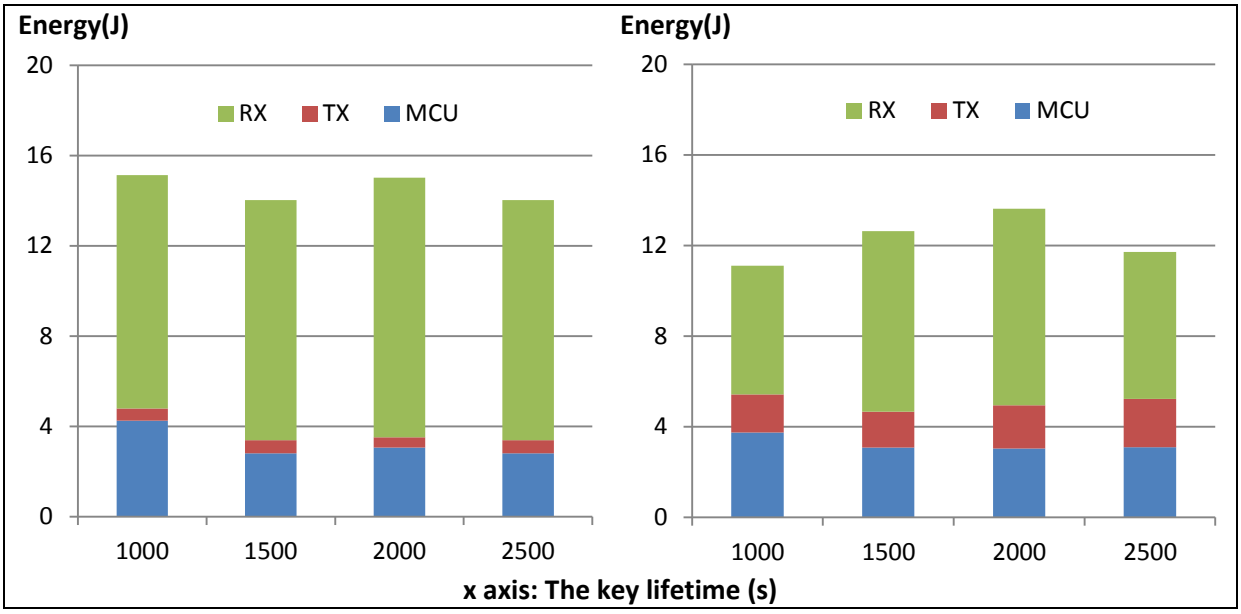


**Figure 4.4**: Energy consumption of (a) the server node "0" and (b) the router node "6" in relation to the key lifetime.

By comparing the energy consumption of every single node as given in the above figures, we can examine how the lifetime of the key affects the sensors' energy consumption and eventually, the performance of the network. Figure 4.4 presents the energy consumption of (a) the server node and

(b) the router node "6" as the key lifetime increases. It can be seen that the energy spent by the micro-processor of the nodes decreases as the key lifetime increases. This happens because the keys are valid for longer time and thus, fewer establishments are needed. Moreover, it can be observed that the energy consumption of the server's receiver is fluctuating, but remains high. This can be explained in brief, as the number of the required key exchanges decrease, the number of the received messages increase and thus the receiver is always busy.

However, the energy consumption analysis which is related to the transmitter and receiver of the router node is probably not very accurate as the routers are not permanently set. The internal routers are defined by the routing protocol of the network and consequently are not the same in every single experiment. Moreover, if a node is set as a router in all of the experiments, the most probable is to serve a different number of clients in every specific simulation. To avoid this inaccuracy we calculate and present in Figure 4.5 the average energy consumption of all the clients, as the lifetime of the key is increased. Figure 4.5 includes data collected by all the nodes except from the server, containing nodes which act as internal routers and nodes acting as clients only.

It can be observed that the average energy consumption of the micro-processor of the clients is clearly decreased as the lifetime of the key increases. The same phenomenon applies also for the energy that is consumed by the transmitter. Actually, a small increase is observed at the value of 1500, but thereafter it follows a downward trend. The average energy is consumed by the receiver of the clients, similar to the transmitter, increase at the lifetime value of 1500 and falls afterwards.

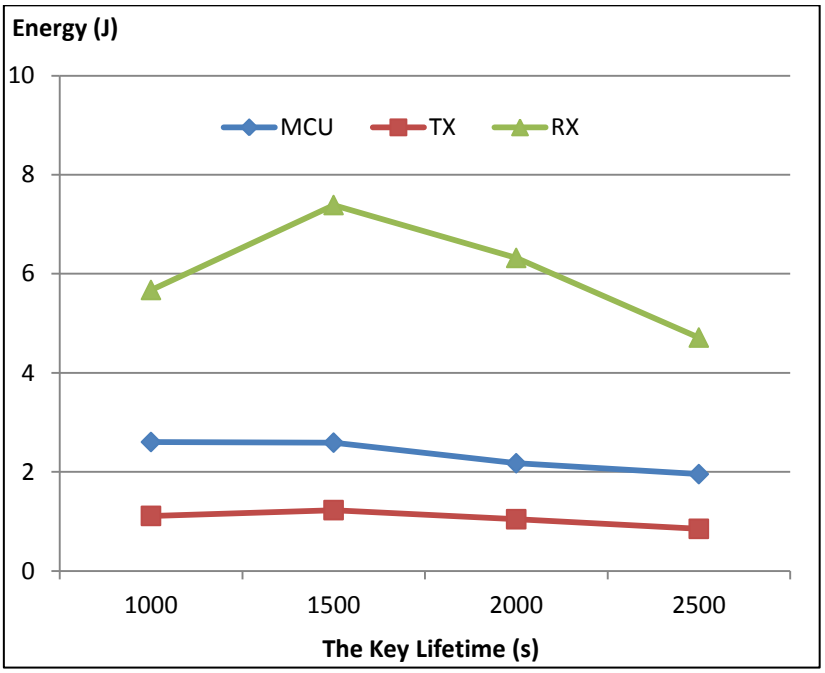


**Figure 4.5**: The average energy consumed by the client nodes.

It is expected that the average energy consumption of the sensor nodes decreases as the key lifetime increases, because the secret keys are valid for longer time and thus less key exchanges are needed. The amount of the energy that the receiver spends is very common for every application running on the sensor nodes. Actually, the receiver normally consumes significantly more energy, but by using the ContikiMAC we reduce the energy consumption dramatically. This happens as the ContikiMAC uses an efficient duty cycle to avoid idle listening of the radio. Thus, we do care more

about the energy spent on the computations and the transmissions, in contrast to the energy consumed by the receiver, as this energy consumption is unavoidable.

Although, it is observed that the energy consumed by the receiver, and thus the overall energy the device spends, is increased at key lifetime of 1500 seconds. This can be explained if we keep in mind the two different cases. For smaller lifetime values the key exchange process is dominating the network and the receiver is mainly used for the purpose of key exchange. As the key lifetime increases the number of the performed key exchanges decrease and consequently the receiver is used for more time by the application layer, listening for incoming packets. The greatest energy consumption by the receiver is observed when the key lifetime has the value of 1500 second.

Actually, when the key lifetime is set to 1500 seconds the usage of the receiver is maximized as its load is balanced between the key exchange processes and the application packet transmission. This happens because the specific value of the key lifetime does not favour the former functionality over the latter, and both of them use the receiver "equally". It is observed that the energy consumed by the receiver, for larger key lifetimes, is decreased as the application layer functionality dominates the device and the key exchanges are performed more rarely. However, the maximum energy consumption is not always observed when the key lifetime is 1500 seconds. Actually, it depends on the size and the setup of the network, but almost in every case the average energy consumption reaches a maximum value and then the decreases. From the specific point onwards (maximum value) the key exchange process is performed less times and the application outbalances the usage of the device components.

Moreover, we investigate how the lifetime of the key affects the overall performance of the network. As explained, the client nodes send messages to the server periodically. When a message moves to the lower layers of the network stack to reach the transmitter, the framer of the MAC layer asks for the secret encryption key. If the key exists the message is forwarded to the transmitter, otherwise it is dropped and the key exchange process starts.

The outgoing packets are not only dropped when a secret key does not exist in the ACL table, but also in the case where the key exchange process is in progress. By deciding to drop the outgoing packets if the communicating nodes do not share a key, the overall performance of the network is affected. However, the existence of the secret key is determined by the lifetime value, which defines if an entry of the ACL table is valid or not. Consequently, the overall performance of the network is affected from the value of the key lifetime.

We have extended the current experiment so that the outgoing messages of the application contain a sequence number, which is unique for the specific node. Every client node tries to send the messages and if a secret key exists the transmission is successful. If a packet is sent over the network successfully by the transmitter, without experiencing any network operational failure, it is properly received by the server's receiver.

To assess how the lifetime of the key affects the performance of the network, we run the current simulation for different lifetime values and count the number of the received messages. The simulation runs over a period of 2 hours. At the end of the simulation we know the number of the messages every single sensor tried to send, because of the sequence number of the last message sent by the application. Moreover, by counting the messages of all the client nodes we are able to determine the overall number of the outgoing packets. On the other hand, the server node counts the received packets, so we are able to estimate the average packet-loss of the network. As it was said, we have simulated the network for different lifetime values to examine how the performance is affected. The percentage of the successfully received packets in relation to the lost packets is presented in Figure 4.6.

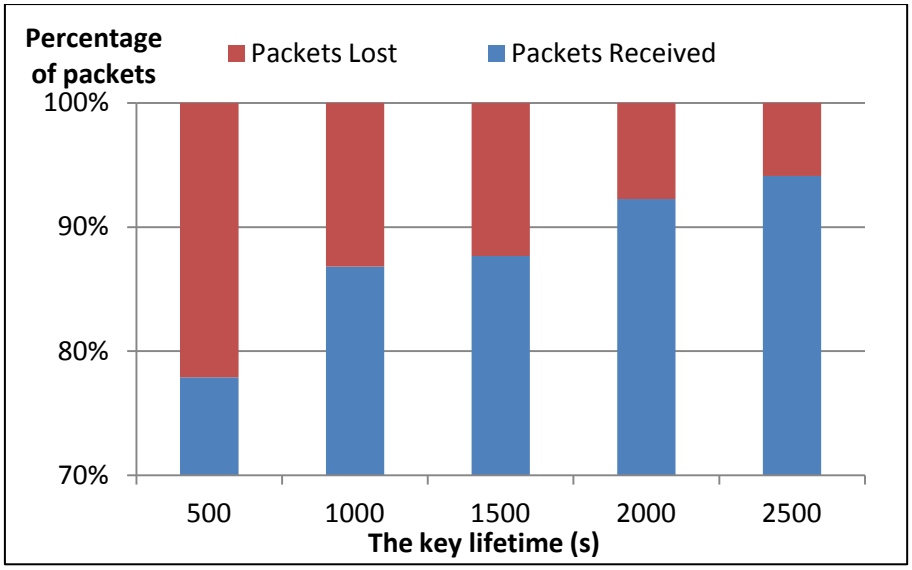


**Figure 4.6**: The packet-loss percentage due to the lack of key in relation to the key lifetime.

As it can be seen in Figure 4.6, when the lifetime of the key is set at the value of 500 seconds, the packets which are lost due to the lack of the key are more than 22 percent of the outgoing packets. This happens because the key lifetime is small and the keys "expire" after a short period of time, and thus the sensor nodes need to establish a large number of secret keys. If a key does not exist or the key exchange process is in progress, all the outgoing packets are dropped. Consequently, we deduce that a large amount of time is spent for the key establishment when the lifetime is small.

Moreover, it is observed that the packet-loss ratio decreases significantly as the key lifetime increases. When the lifetime is set to 1000 and 1500 the packet-loss of the networks is 13% and 12% respectively and eventually, when the lifetime is at least 2000 seconds the packet-loss gets lower than 10 percent. More specifically, at the value of 2500 seconds only 5% of the packets are dropped due to the lack of a key.

With the current experiment we conclude that as the lifetime of the secret keys increases, the number of lost packets decreases and that the network provides better performance. We have also seen how the key lifetime affects the energy consumption of the sensor nodes when the micro-processor has the duty to perform the elliptic curve multiplications. Eventually, if the lifetime is small, all the effort is paid for the key exchange process and not for packet transmissions.

It is recommended that the key lifetime is set to a large value to avoid dropping the outgoing packets or blocking the transmission by having continuous key exchanges. Actually, the lifetime of the key is related to the size of the network and thus it must increase as the network size increases. This is expected as the nodes need to establish and keep a valid key to be able to communicate. For these two reasons we recommend that the lifetime value should be at least 2500 seconds for a network of 10-15 nodes and even greater for larger networks.

However, if the nodes are assigned to the network gradually and the key exchange operations are not triggered at the same time, the provided performance will be better. The major problem is that all the nodes start the key exchange at the same time, after the initialisation phase of the network, and all the key re-establishments roughly coincide at a later stage. Therefore, if the different key exchanges are performed on fixed but random times, we will avoid the situation where a node is blocked because of waiting for key exchange, while the opponent node is unavailable as it currently is in progress.

## 4.4 Experiment: Scalability analysis of the network

In this experiment we try to address the scalability of the 6LowPAN networks and how it is affected by the implemented key exchange process. The default size of the ACL table was set to 4 for all the previous simulations, which means that every sensor node is able to keep 4 address-key pairs in the association table. If a new key is needed, we have implemented a simple solution to erase the oldest entry of the table, and use the free space to store the new key. In the current experiment we change the size of the ACL table to check how the network responds and its scalability.

We gradually increase the size of the table and examine the memory overhead of each device by observing the size of the RAM and ROM memory that is consumed. The basic observation is that every sensor node is able to have up to 36 entries in its ACL table without facing any problem. The amount of memory that is consumed by the nodes as the size of the ACL table increases is presented in Figure 4.7.

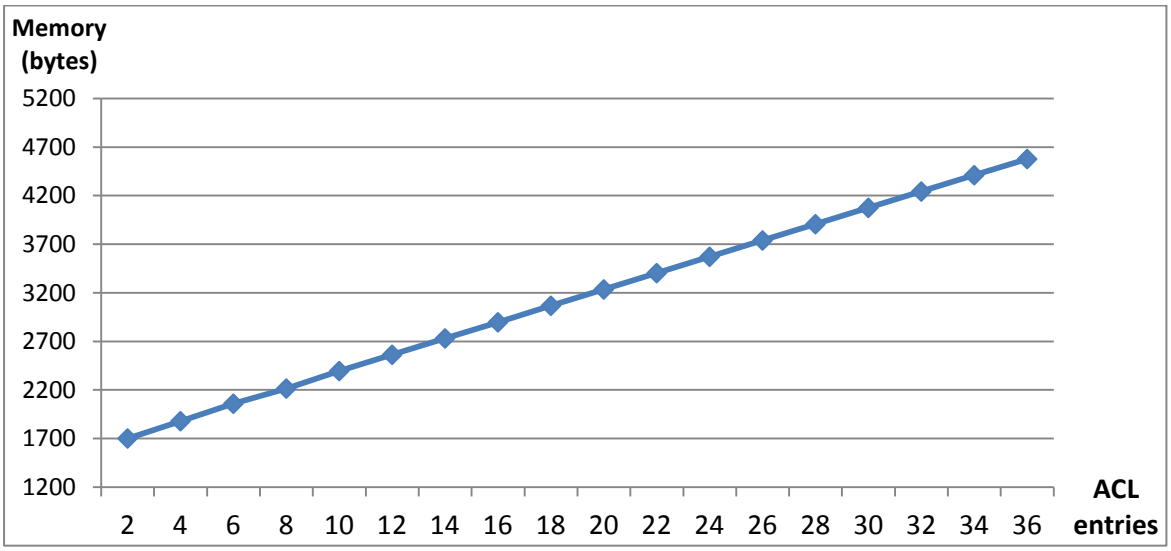


**Figure 4.7**: Memory consumption in relation to the ACL size.

As mentioned, every node is able to store 36 secret keys by having the corresponding entries in its table. If we define the ACL table with larger size, in order to keep a larger number of entries, the compiler/linker crashes as we try to use more RAM memory than the available. Actually, we observe that the memory requirements increase linearly as the number of the keys becomes larger. Specifically, we know that every association of the ACL requires 84 bytes of RAM memory.

The presented results do not mean that the network is only able to support 36 nodes, but that every sensor node is able to store 36 different secret keys. By knowing that the associations of the ACL table are replaced if it is requested and that hop-by-hop routing is performed, we conclude that the network is very scalable. Actually, the sensor nodes do not need to establish secret keys and direct communication links with all of the other nodes, but only with their neighbors, which perform the routing operations.

As the network is multi-hop, it is necessary for a sensor node only to establish communication links and secret keys with some of its nearest neighbors. It follows that the network by supporting up to 36 associations per device becomes of very large scale if its density is low. Eventually, it can be said that the size of the key table is quite large to support the existence of a very large number of nodes and thus the networks can easily be extended.

# 5. CONCLUSIONS

As stated in the introduction chapter, the scope of the current project is to investigate whether any of the existing key exchange techniques for the WSNs is applicable to the 6LowPAN networks, which are IPv6-based and lossy wireless sensor networks. This aim stems from the fact that we need a reliable solution to guarantee security for the link layer frames by providing secret encryption keys. Actually, link layer security is essential for 6LowPAN networks as the specific networks provide multi-hop communication by performing the necessary routing operations. More specifically, the packets are not sent directly, but routed through multiple internal nodes to reach their destination.

In order to be able to provide link layer security guarantees we have studied the IEEE 802.15.4 specification standard, which defines the link layer characteristics of the 6LowPAN networks. Moreover, we studied how IEEE 802.15.4 standard supports link layer security and thus we have implemented the auxiliary security header in the Contiki OS, as specified by the standard. This enables the transmission of secure frames, if requested by the application and a secret key exists. Moreover, we have modified the creating and parsing functions of the link layer framer in order to be able to properly construct and parse the secure frames. By this, the objective of providing link layer guarantees has been achieved.

However, the aim of the project is not simply to provide link layer guarantees but to investigate if any of the existing key exchange techniques for the traditional WSNs can be applied to the 6LowPAN networks. To fulfil the scope of the project we researched on a number of different key exchange techniques that are currently used in the traditional WSNs. These techniques fall into two major categories, where the former consists of techniques that use a key distribution centre, preloaded keys and the pool scheme and the latter consists of techniques that use the public key cryptography. By the research done on the specific scientific area, we have concluded that the most promising technique is based on elliptic curve cryptography. For this reason we have decided to use elliptic curves to implement the ECDH technique on the Contiki OS.

The decision to utilise elliptic curves was correctly taken as they provide the good characteristics of the public-key cryptography along with minimised time and energy consumption. The advantages of ECC stem from the fact that the keys have small size in comparison to other techniques. Moreover, we have decided not to permanently store the private and the public keys, but to discard them and create a new pair every time a key exchange is requested. As mentioned earlier, this decision leads to larger time and energy consumption in contrast to the case where the keys are permanently stored. However, the provided security is stronger as the private and public keys cannot be leaked (they are not stored). Additionally, this decision allows implementing a lifetime mechanism for the secret keys and thus a leakage of a key affects only a pair of nodes for a small time period (as long as the specific key is valid).

The first objective of the project is to implement in the Contiki OS the key exchange technique that seems most suitable for 6LowPAN networks, in order to evaluate its applicability, viability and scalability for large scale IPv6 networks. This objective has been achieved as we managed to use elliptic curve cryptography to implement the ECDH algorithm. We have developed a complete solution of the proposed key exchange algorithm, not only establishing a secret key between two nodes, but also providing solutions on how the keys are stored, searched, replaced and deleted.

By constructing the ACL table for every node of the network, we allow each node to store a default pre-defined number of keys, which are shared with its neighbors. Moreover, we have implemented a mechanism that defines a lifetime value for every key. The lifetime is assigned at the key creation time and is decreased periodically. When the lifetime reaches zero, the key association is erased from the ACL table. Also, if all the associations of the ACL are valid but a new one is requested, we manage to find and replace the oldest of the existing associations. With the current implementation

we conclude that the proposed key exchange technique is applicable for the 6LowPAN networks and that it can easily be used over the Internet stack, with the support of UDP/IP communication. Moreover, ECDH requires a small amount of memory, and thus is suitable for devices with limited resources.

Moreover, the second objective is to run a number of different experiments on the Cooja simulator in order to assess its viability, its scalability and the performance of the network when the key exchange is enabled. After implementing the ECDH key exchange technique we have designed and built a number of different and specifically targeted simulation experiments. In these simulations we measure the time the elliptic curve point multiplication operation takes and the energy consumption as well. Not only that, but also we examine and analyze the overall amount of time and energy that is consumed for a key exchange to be completed. Moreover, we run some simulations to address how the key lifetime affects the performance of the network, the packet-loss ratio and the energy consumption of the devices.

By examining and analyzing the experimental results we conclude that the ECHD key exchange technique is a viable solution for 6LowPAN networks. The experimental results show that the sensor devices consume an acceptable amount of time and energy to perform the key exchange process. Moreover, we observe that the impact of the key exchange process on the overall performance of the network is limited. It was presented that the packet-loss ratio of the network as well as the energy consumption of the devices is affected by the lifetime of the secret keys. Consequently, we suggest that the lifetime of the established keys must be set to a large value (more than 2000 seconds) to avoid having large computational overhead by performing a sequence of key exchanges continuously. Not only that, but also the energy consumption of the devices is related to the size and the setting of the network, as well as the key lifetime value. It is suggested that a larger lifetime value must be set for large-scale and dense networks.

Also we have set as our objective to test whether the chosen key exchange technique affects the scalability of the 6LowPAN networks. We observed through the experiments that the network is very scalable because it allows routing and hop-by-hop communication as well as that every node is able to keep up to 36 distinct keys in its ACL table. This is a very large number if we keep in mind that each sensor node needs to establish keys only with its neighbors and that the communication is performed by routing and by multiple hops.

To achieve the objectives we have set in this project, we researched the current key exchange techniques and we decided to implement the ECDH. By implementing it, testing and evaluating it, we conclude that the specific technique is applicable for 6LowPANs. The ECDH technique is viable for 6LowPANs, providing a good performance over the Internet stack. Moreover, we conclude that the ECDH implementation is very scalable and thus suitable for large-scale networks. By achieving the set objectives we fulfil the scope of the project.

# 6. FUTURE WORK

In the current project we have implemented the ECDH key exchange technique, which is based on elliptic curve cryptography. The elliptic curve we have used in the current project is the standardized SECP128R1 curve, where its points have size 128-bits. We plan to use some different elliptic curves, of size 160 and 192 bits as well, to examine their memory footprint as well as the time and energy consumption. Moreover, we will assess how the size of the curve affects the performance of the network.

Currently, the nodes of the network communicate and establish a secret shared key, which is destined to be used for AES encryption. We have run and tested a software implementation of the AES algorithm, which works properly as a standalone component, but we do not to use it at the moment as it requires about 4k of memory. We plan to implement the AES algorithm in hardware where it would be faster and avoid consuming memory. After employing AES we will run a number of different experiments to assess how the encryption of the packets affects the performance of the network. It is also interesting to examine the requirements and the efficiency of the overall solution.

It is very important to secure the sensor nodes against the man-in-the-middle attack, at the key exchange phase. We can avoid the man-in-the-middle attack by employing the ECDSA, which allows the nodes to be mutually authenticated or by using ECMQV for the key exchange. We already have an implementation of the ECDSA algorithm (ContikiECC project), but a slight modification is needed in order to be combined and integrated with the current key exchange solution. Actually, the ECDSA solution currently stands as an application, where we have to implement it working over the link layer. Moreover, we must investigate and test if it is possible to reduce the memory footprint of the ECDSA. After that, some experiments are required to assess the performance of the network, to examine and analyse the overall resource requirements. It would also be nice to implement the ECMQV and compare its performance to the simple ECDH solution with the use of ECDSA.

The solution provided in the current project does not allow the negotiating nodes to concurrently compute their public key, while the former node computes its own and sends it to the latter as a key exchange request. It is important to examine the case where the two sensor nodes compute their public keys simultaneously and exchange them after the computation. By this, the overall time Alice consumes for key exchange is reduced, because she does not remain idle waiting for Bob's public key. The disadvantage of this approach is that the traffic of the network increases by sending synchronisation messages. It is important to investigate if this method provides better overall performance in comparison to the implemented solution.

The current key exchange technique is tested only with simulation experiments on the Cooja simulator, and due to the lack of time it was not tested or real hardware devices. It is important to evaluate the ECDH key exchange by performing real hardware experiments with the available devices. Finally, the future work would be focused on different elliptic curve optimisations, which can be implemented on resource-constraint devices. By researching the different optimizations we are able to reduce the time and energy consumption. A significant research has been already done in this area, but the proposed techniques have to be implemented and tested for 6LowPAN networks.

# BIBLIOGRAPHY


[1] "contikiecc - Contiki port of TinyECC." [Online]. Available: http://score.ucsc.lk/projects/contikiecc. [Accessed: 22-Jun-2012].

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

[3] C. P. P. Schumacher, N. Kushalnagar, and G. Montenegro, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals." [Online]. Available: http://tools.ietf.org/html/rfc4919. [Accessed: 17-Sep-2012].

[4] L. Tan and N. Wang, "Future internet: The Internet of Things," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, 2010, vol. 5, pp. 376 –380.

[5] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[6] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, Washington, DC, USA, 2004, pp. 455–462.

[7] F. Österlind, *A Sensor Network Simulator for the Contiki OS, SICS technical report*. Swedish Institute of Computer Science, 2006.

[8] "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), IEEE Standard, 802.15.4-2006," 2006.

[9] J. T. Adams, "An introduction to IEEE STD 802.15.4," in *Aerospace Conference, 2006 IEEE*, 2006, p. 8 pp.

[10] M. S. Hossen, A. F. M. S. Kabir, R. H. Khan, and A. Azfar, "Interconnection between 802.15.4 Devices and IPv6: Implications and Existing Approaches," *arXiv:1002.1146*, Feb. 2010.

[11] N. Sastry and D. Wagner, "Security considerations for IEEE 802.15.4 networks," in *Proceedings of the 3rd ACM workshop on Wireless security*, New York, NY, USA, 2004, pp. 32–42.

[12] L. Casado and P. Tsigas, "ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System," presented at the Proceedings of the 14th Nordic Conference on Secure IT Systems (NordSec 2009), Lecture Notes in Computer Science, 2009, vol. 5838, pp. 133–147.

[13] M. Baar, E. Koppe, A. Liers, and J. Schiller, "Poster abstract: The scatterweb msb-430 platform for wireless sensor networks," in *Contiki Workshop'07*, 2007.

[14] I. Akyildiz and M. C. Vuran, *Wireless Sensor Networks*. New York, NY, USA: John Wiley & Sons, Inc., 2010.

[15] Z. Shelby and C. Bormann, *6LoWPAN: The Wireless Embedded Internet*. Wiley Publishing, 2010.

[16] K. H. Kim, W. Haddad, J. Laganier, S. Park, and S. Chakrabarti, "IPv6 over Low Power WPAN Security Analysis." [Online]. Available: http://tools.ietf.org/html/draft-daniel-6lowpan-security-analysis-01. [Accessed: 05-May-2012].

[17] R. Riaz, K. H. Kim, and H. F. Ahmed, "Security analysis survey and framework design for IP connected LoWPANs," in *Autonomous Decentralized Systems, 2009. ISADS '09. International Symposium on*, 2009, pp. 1 –6.

[18] S. Raza, T. Voigt, and U. Roedig, "6LoWPAN Extension for IPsec.," presented at the Interconnecting Smart Objects with the Internet Workshop, Prague, Czech Republic, 2011.

[19] S. Bac, D. Kwak, and C. Kim, “Information Networking. Towards Ubiquitous Networking and Services,” T. Vazão, M. M. Freire, and I. Chong, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 142–150.

[20] A. Dunkels, L. Mottola, N. Tsiftes, F. Österlind, J. Eriksson, and N. Finne, “The announcement layer: beacon coordination for the sensornet stack,” in *Proceedings of the 8th European conference on Wireless sensor networks*, Berlin, Heidelberg, 2011, pp. 211–226.

[21] F. Wang and J. Liu, “Duty-Cycle-Aware Broadcast in Wireless Sensor Networks,” in *INFOCOM 2009, IEEE*, Rio de Janeiro, 2009, pp. 468 –476.

[22] S. Guo, Y. Gu, B. Jiang, and T. He, “Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links,” in *Proceedings of the 15th annual international conference on Mobile computing and networking*, New York, NY, USA, 2009, pp. 133–144.

[23] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos, “Key management systems for sensor networks in the context of the Internet of Things,” *Computers & Electrical Engineering*, vol. 37, no. 2, pp. 147–159, Mar. 2011.

[24] J. Zhang and V. Varadharajan, “Review: Wireless sensor network key management survey and taxonomy,” *J. Netw. Comput. Appl.*, vol. 33, no. 2, pp. 63–75, Mar. 2010.

[25] J. Kohl and C. Neuman, “The Kerberos Network Authentication Service (V5),” Jan. 1993.

[26] C. Lederer, R. Mader, M. Koschuch, J. Großschädl, A. Szekely, and S. Tillich, “Energy-Efficient Implementation of ECDH Key Exchange for Wireless Sensor Networks,” in *Proceedings of the 3rd IFIP WG 11.2 International Workshop on Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, Berlin, Heidelberg, 2009, pp. 112–127.

[27] H. Chan, A. Perrig, and D. Song, “Random key predistribution schemes for sensor networks,” in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, 2003, pp. 197 – 213.

[28] L. Eschenauer and V. D. Gligor, “A key-management scheme for distributed sensor networks,” in *Proceedings of the 9th ACM conference on Computer and communications security*, New York, NY, USA, 2002, pp. 41–47.

[29] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, “Recommendation for key management–part 1: General (revision 3),” *NIST special publication*, vol. 800, p. 57, 2011.

[30] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[31] H. Wang and Q. Li, “Efficient Implementation of Public Key Cryptosystems on Mote Sensors,” in *Information and Communications Security*, vol. 4307, P. Ning, S. Qing, and N. Li, Eds. Springer Berlin / Heidelberg, 2006, pp. 519–528.

[32] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs,” in *Cryptographic Hardware and Embedded Systems - CHES 2004*, vol. 3156, M. Joye and J.-J. Quisquater, Eds. Springer Berlin / Heidelberg, 2004, pp. 925–943.

[33] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, “Energy analysis of public-key cryptography for wireless sensor networks,” in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, 2005, pp. 324 – 328.

[34] W. Diffie, P. C. Oorschot, and M. J. Wiener, “Authentication and authenticated key exchanges,” *Designs, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[35] S. Burnett and S. Paine, *The RSA Security’s Official Guide to Cryptography*. New York, NY, USA: McGraw-Hill, Inc., 2001.

[36] “Elliptic Curve: A Geometric Approach.” [Online]. Available: http://www.certicom.com/index.php/21-elliptic-curve-addition-a-geometric-approach. [Accessed: 19-Jul-2012].
[37] J. M. Pollard, “Monte Carlo Methods for Index Computation (mod p),” *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924, Jul. 1978.
[38] S. Pohlig and M. Hellman, “An improved algorithm for computing logarithms over GF(p) and its cryptographic significance,” *Information Theory, IEEE Transactions on*, vol. 24, no. 1, pp. 106 – 110, Jan. 1978.
[39] P. Oorschot and M. Wiener, “Parallel Collision Search with Cryptanalytic Applications,” *Journal of Cryptology*, vol. 12, no. 1, pp. 1–28, 1999.
[40] A. Lenstra, H. Lenstra, M. Manasse, and J. Pollard, “The number field sieve,” in *The development of the number field sieve*, vol. 1554, A. Lenstra and H. Lenstra, Eds. Springer Berlin / Heidelberg, 1993, pp. 11–42.
[41] P. Gaudry, F. Hess, and N. Smart, “Constructive and destructive facets of Weil descent on elliptic curves,” *Journal of Cryptology*, vol. 15, no. 1, pp. 19–46, 2002.
[42] D. J. Malan, M. Welsh, and M. D. Smith, “Implementing public-key infrastructure for sensor networks,” *ACM Trans. Sen. Netw.*, vol. 4, no. 4, pp. 22:1–22:23, Sep. 2008.
[43] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone, “An Efficient Protocol for Authenticated Key Agreement,” *Designs, Codes and Cryptography*, vol. 28, no. 2, pp. 119–134, 2003.
[44] J. Großschädl, A. Szekely, and S. Tillich, “The Energy Cost of Cryptographic Key Establishment in Wireless Sensor Networks,” in *PROCEEDINGS OF THE 2ND ACM SYMPOSIUM ON INFORMATION, COMPUTER AND COMMUNICATIONS SECURITY (ASIACCS 2007)*, 2007, pp. 380–382.
[45] P. Rogaway, M. Bellare, and D. Boneh, “Evaluation of security level of cryptography: ECMQVS (from SEC 1).” CRYPTREC report, Information-technology Promotion Agency, Jan-2001.
[46] K. Igoe, D. McGrew, and M. Salter, “Fundamental Elliptic Curve Cryptography Algorithms.” [Online]. Available: https://art.tools.ietf.org/html/rfc6090. [Accessed: 10-May-2012].
[47] H. Wang, B. Sheng, C. C. Tan, and Q. Li, “Public-key based access control in sensornet,” *Wireless Networks.*, vol. 17, no. 5, pp. 1217–1234, Jul. 2011.
[48] “TinyOS.” [Online]. Available: http://www.tinyos.net/. [Accessed: 01-Sep-2012].
[49] Wikipedia contributors, “Elliptic curve point multiplication,” *Wikipedia, the free encyclopedia*. Wikimedia Foundation, Inc., 30-Aug-2012.
[50] H. F. Huang, “A New Design of Efficient Key Pre-distribution Scheme for Secure Wireless Sensor Networks,” in *Intelligent Information Hiding and Multimedia Signal Processing, 2007. IIHMSP 2007. Third International Conference on*, 2007, vol. 1, pp. 253 –256.
[51] P. Montgomery, “Modular Multiplication without Trial Division,” *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.
[52] H. Cohen, A. Miyaji, and T. Ono, “Efficient Elliptic Curve Exponentiation Using Mixed Coordinates,” in *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, London, UK, UK, 1998, pp. 51–65.
[53] G. Bianchi, A. T. Capossele, A. Mei, and C. Petrioli, “Flexible key exchange negotiation for wireless sensor networks,” in *Proceedings of the fifth ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, New York, NY, USA, 2010, pp. 55–62.

[54] R. Mzid, M. Boujelben, H. Youssef, and M. Abid, “Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography,” in *Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS), 2010 International Conference on*, 2010, pp. 1 –8.

[55] W. Jung, S. Hong, M. Ha, Y. J. Kim, and D. Kim, “SSL-Based Lightweight Security of IP-Based Wireless Sensor Networks,” in *Advanced Information Networking and Applications Workshops, 2009. WAINA ’09. International Conference on*, 2009, pp. 1112 –1117.

[56] A. Sahana and I. S. Misra, “Implementation of RSA security protocol for sensor network security: Design and network lifetime analysis,” in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, 2011, pp. 1 –5.

[57] “Contiki: The Open Source Operating System for the Internet of Things.” [Online]. Available: http://www.contiki-os.org/#about. [Accessed: 01-Sep-2012].

[58] T. W. <wintert@acm.org>, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.” [Online]. Available: http://tools.ietf.org/html/draft-ietf-roll-rpl-06. [Accessed: 03-Sep-2012].

[59] A. Dunkels, “The contikimac radio duty cycling protocol,” *SICS technical report, Swedish Institute of Computer Science*, 2011.

[60] “Tmote Sky.” [Online]. Available: http://www.snm.ethz.ch/Projects/TmoteSky. [Accessed: 01-Sep-2012].

[61] “MSP430 Microcontroller | 16 bit Microcontroller | 16 bit MCU – TI.com.” [Online]. Available: http://www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page. [Accessed: 19-Sep-2012].

[62] “Libelium - Wireless Sensor Networks - ZigBee - Mesh Networks.” [Online]. Available: http://www.libelium.com/security_802.15.4_zigbee/. [Accessed: 19-Sep-2012].

[63] N. Tsiftes, J. Eriksson, and A. Dunkels, “Low-power wireless IPv6 routing with ContikiRPL,” in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, New York, NY, USA, 2010, pp. 406–407.

[64] “AES128 – A C Implementation for Encryption and Decryption (Rev. A).” [Online]. Available: http://www.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=slaa397a. [Accessed: 02-Jul-2012].

[65] A. Liu and P. Ning, “TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks,” in *in Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN 2008*, 2008, pp. 245–256.

[66] “Tmote Sky Datasheet,” 2006 [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf. [Accessed: 01-Sep-2012].