UNIVERSITÀ DI PISA

Scuola di Dottorato in Ingegneria "Leonardo da Vinci"

Corso di Dottorato di Ricerca in
INGEGNERIA DELL'INFORMAZIONE

Tesi di Dottorato di Ricerca

Performance Evaluation of Security Solutions for Wireless Sensor Networks

Autore:

Roberta Daidone ______________________

Relatore:

Prof. Gianluca Dini ____________________

Anno 2014

A me stessa.
A chi c’è sempre stato.
A chi è arrivato giusto in tempo.

Sommario

Negli ultimi anni, le comunicazioni wireless hanno iniziato a coinvolgere non solo i computer, ma una grande quantità di oggetti eterogenei. Le *Wireless Sensor Network* (*WSN*) contribuiscono al nuovo paradigma del *pervasive computing*, traducendosi in nuovi requisiti per nuove applicazioni. Le WSN sono impiegate non soltanto indipendentemente, ma anche nei *Cooperating Objects System* (*COS*), dove diversi agenti fisici mobili condividono l'ambiente operativo per perseguire uno scopo comune.

I nodi sensori tipicamente possiedono risorse limitate e vengono posti in zone ostili. WSN e COS sono un obiettivo invitante per un avversario, in quanto una violazione della sicurezza delle comunicazioni può facilmente tradursi in una violazione della sicurezza del sistema, con il rischio di danneggiare persone o cose. I principali requisiti di sicurezza per le WSN sono le comunicazioni sicure, la gestione delle chiavi crittografiche e il bootstrap sicuro. In genere, la sicurezza comporta operazioni pesanti dal punto di vista computazionale, mentre i sensori dispongono di risorse limitate. Ciò significa che i requisiti di sicurezza di una WSN vanno soddisfatti assicurando un impatto leggero sulle prestazioni, in termini di occupazione di memoria, carico della rete e consumo energetico.

Questo lavoro di tesi parte da una valutazione delle prestazioni della sicurezza dello standard IEEE 802.15.4, in termini di occupazione di memoria, prestazioni della rete e consumo energetico. Poi viene presentata la soluzione a una vulnerabilità dello standard IEEE 802.15.4, che causa un attacco di *Denial of Service* selettivo. Infine, si presenta *PLASA*: un'architettura di sicurezza per reti di sensori modulare e riconfigurabile. PLASA estende l'architettura di *STaR*. STaR è un modulo per comunicazioni sicure da noi progettato per garantire confidenzialità e/o autenticità delle comunicazioni in maniera trasparente e flessibile. PLASA parte dal nucleo di STaR, introducendo i moduli per la gestione delle chiavi e per il bootstrap sicuro, così da fornire un sistema fruibile non solo per le WSN, ma per i COS nella loro interezza.

Abstract

In the recent years, wireless communication is involving not only computers, but a multitude of heterogeneous devices. *Wireless Sensor Networks* (*WSNs*) contribute to the new paradigm of *pervasive computing*, and this translates into new requirements for new applications. WSNs are employed not only on their own, but also in *Cooperating Objects Systems* (*COSs*), where mobile physical agents share the same environment to fulfill their tasks, either in group or in isolation.

Sensor nodes are typically resource constrained devices deployed in unattended, possibly hostile environments. WSNs and COSs are a tempting target for an adversary, since a security infringement may easily translate into a safety one, with possible consequences in terms of damages to things and injures to people. Main security requirements for WSNs are secure communication, key management and secure bootstrapping. Security usually involves resource greedy operations, while sensors are resource constrained devices. This means that security requirements must be satisfied assuring a lightweight impact in terms of memory occupancy, network performance and energy consumption.

In this thesis work, we start from a performance evaluation of the security sublayer of the *IEEE 802.15.4* standard in terms of memory occupancy, network performance and energy consumption. Then, present and evaluate a solution to a vulnerability of the IEEE 802.15.4 standard that causes a selective *Denial of Service* attack. Finally, we present *PLASA*: a modular and reconfigurable security architecture for WSNs. PLASA extends the *STaR* architecture. STaR is a secure communication module we designed to provide confidentiality and/or authenticity of communications in a transparent and flexible manner. PLASA enhances STaR, introducing modules for key management and secure bootstrapping, so providing a complete system that is suitable not only for the WSN, but for the entire COS.

Contents

1 Introduction . . . 1

2 The IEEE 802.15.4 security sublayer . . . 3

2.1 Related work . . . 5

2.2 IEEE 802.15.4: an overview . . . 6

2.2.1 IEEE 802.15.4 security sublayer . . . 8

2.2.2 Security operations . . . 11

2.2.3 The CONET open implementation of IEEE 802.15.4 . . . 12

2.3 Evaluation . . . 13

2.3.1 Analysis . . . 13

Latency and goodput . . . 14

Per-packet energy consumption . . . 15

Evaluation of parameters . . . 16

Analytical results . . . 18

Experimental validation of the analytical model . . . 22

2.3.2 Simulation analysis . . . 23

2.3.3 Experimental evaluation of memory overhead . . . 28

3 Selective Denial of Service in IEEE 802.15.4 networks . . . 31

3.1 Wireless Denial of Service: the jamming attack . . . 32

3.2 IEEE 802.15.4 MAC overview . . . 34

3.2.1 Guaranteed Time Slot (GTS) . . . 36

3.2.2 Security services . . . 38

3.3 GTS-based selective jamming attack . . . 39

3.4 Selective Jamming Resistant GTS (SJRG) . . . 40

3.4.1 Discussion on dictionary attack . . . 42

3.5 SJRG implementation . . . 43

3.6 SJRG evaluation . . . 44

3.6.1 Effectiveness . . . 44

3.6.2 Memory footprint . . . 45

3.6.3 Network performance 46
3.6.4 Energy consumption 48
3.6.5 On scalability 49

4 PLASA 51
4.1 Related work 52
4.2 PLASA architecture 53
4.2.1 STaR module 54
STaR security services 56
STaR communication support 57
STaR dynamic reconfiguration 58
4.2.2 Authentication module 60
4.2.3 KeyDB module 61
4.2.4 Key management module functions 61
4.3 PLASA performance evaluation 62
4.3.1 Memory occupancy 62
4.3.2 PLASA communication and processing overhead 64
4.3.3 PLASA energy consumption 66
4.4 PLASA integration study case 67
4.4.1 PLASA to protect a COS from scratch 68
PLASA integration 71

5 Conclusion 73

References 75

List of Figures

2.1 IEEE 802.15.4 superframe structure. 7
2.2 Auxiliary Security Header (ASH) structure. 9
2.3 CTR security level. 9
2.4 CBC-MAC security level. 9
2.5 CCM security level. 9
2.6 Format of ASH in KeyIdMode0. 10
2.7 Format of ASH in KeyIdMode1. 11
2.8 Format of ASH in KeyIdMode2. 11
2.9 Format of ASH in KeyIdMode3. 11
2.10 Latency timeline without security. 14
2.11 Latency timeline with security. 14
2.12 Latency for HW-based cryptography with different security levels. 19
2.13 Goodput for HW-based cryptography with different security levels. 19
2.14 Energy for HW-based cryptography with different security levels. 20
2.15 Latency for SW-based cryptography with different security levels. 20
2.16 Goodput for SW-based cryptography with different security levels. 21
2.17 Energy for SW-based cryptography with different security levels. 21
2.18 Simulated latency for HW-based cryptography. 24
2.19 Simulated goodput for HW-based cryptography. 25
2.20 Simulated energy consumption for HW-based cryptography. 25
2.21 Simulated latency for SW-based cryptography. 27
2.22 Simulated goodput for SW-based cryptography. 27
2.23 Simulated energy consumption for SW-based cryptography. 28
2.24 ROM memory overhead with HW-based cryptography. 29
2.25 ROM memory overhead with SW-based cryptography. 29
2.26 RAM memory overhead. 30

3.1 Beacon frame format. 35
3.2 IEEE 802.15.4 superframe structure. 35

3.3 GTS allocation and deallocation requests. 36
3.4 GTS request command format. 37
3.5 Format of the GTS fields. 37
3.6 Format of the GTS descriptor. 38
3.7 Example of GTS-based selective jamming attack. 39
3.8 SJRG memory occupancy on Tmote Sky motes. 46

4.1 PLASA component overview. 53
4.2 Example of packet processed by STaR. 55
4.3 STaR architecture. 55
4.4 Outgoing packet processing. 58
4.5 Incoming packet processing. 58
4.6 STaR control packet format. 59
4.7 PLASA memory footprint. 63
4.8 PLASA secured COS scenario. 68

List of Tables

2.1 Security levels. 10
2.2 Size of the ASH as a function of the KeyIdMode. 11
2.3 Parameters. 17
2.4 Frame expansion due to security. Values are in bytes. 18
2.5 Experimental vs analytical latency. Values are in ms. 23
2.6 Experimental vs analytical goodput. Values are in Kbit/s. 23
2.7 Simulative vs. experimental latency and goodput with HW-based encryption. 26

3.1 List of acronyms. 34
3.2 SJRG processing overhead contributions. 47
3.3 SJRG transmission overhead contributions. 48
3.4 SJRG energy consumption contributions. 48

4.1 Detailed memory occupancy. 62
4.2 PLASA $\mathrm{d}_{\mathrm{proc}}$ contributions for PLASA modules. 65
4.3 PLASA d_{tx} contributions for PLASA modules. 65
4.4 PLASA energy consumption contributions. 67

1 Introduction

In the recent years, *Wireless Sensor Networks* (*WSNs*) have received an increasing amount of attention and have been adopted in many application scenarios, from environmental to health care monitoring applications. In such scenarios, sensor nodes collect environmental data, and transmit them to a central base station through a wireless network. WSNs have been used chiefly for scientific purposes, where an adversary has little incentive to attack sensors [1].

Sensor nodes are typically resource constrained devices deployed in unattended, possibly hostile environments. This implies that devices are exposed to the risk of being compromised. In order to avoid this, security should be introduced in WSNs. Introducing security in WSNs increases the demand for computational resources and memory to store temporary data. These extra requirements may become an issue because sensors are resource constrained devices, with scarce computational power and shortage of memory. Also, sensors are usually battery powered, and security operations have an impact also on battery lifetime.

In order to provide the above mentioned applications with secure communication, we recommend to keep into account the following issues.

- Evaluate security costs in advance to trade-off sensors performance and security. Once security costs are known, it is possible to choose the best solution in terms of both security and network sustainability.
- Rely on modular and flexible security architectures, so making it possible to choose the best solution in terms of both security and network sustainability.
- Design mechanisms that provide different security policies and allow to switch from a policy to another, assuring high adaptation to changes in the network.

In this Ph.D. dissertation, we consider the above mentioned assertions about security in WSNs, and support their foundation by means of the following contributions.

In Chapter 2 we present a thorough investigation of costs of the IEEE 802.15.4 security sublayer. We have evaluated the costs of different security settings provided by the IEEE 802.15.4 standard and cross-validated them by means of analysis, simulations and real experiments. Real experiments result in a twofold contribution: they provide validation of analysis and simulations, but also allow to evaluate the impact of security on memory occupancy. Realizing our own IEEE 802.15.4 security sublayer implementation [2] for TinyOS [3] and Tmote Sky motes [4], gave us the knowledge of constraints in the number of Tmote Sky motes in a PAN when using security.

In Chapter 3 we complete the work of Sokullu *et al.* [5, 6] enlarging the range of possible GTS-based selective jamming incarnations with the *sniper attack*. Also, we present an IEEE 802.15.4 standard-compliant countermeasure against the GTS-based selective jamming attack, i.e. *Selective Jamming Resistant GTS* (*SJRG*). SJRG is based on two basic mechanisms: i) protection of secrecy and integrity of beacon and GTS request frames by means of encryption; and ii) protection from network traffic analysis by means of intra-Slot randomization. While several solutions can be devised, the challenge is to devise one that is compliant with the IEEE 802.15.4 standard. We took up the challenge and conceived the aforementioned mechanisms in such a way that SJRG is fully compliant with IEEE 802.15.4.

In Chapter 4 we present PLASA, a modular security suite to provide WSNs with secure communications, key management and secure bootstrapping. PLASA includes STaR, a modular, reconfigurable and transparent software component for secure communications in WSNs. STaR guarantees confidentiality, integrity, and authenticity by means of encryption and/or authentication. STaR is *modular* because it separates interfaces from their implementations. STaR is *reconfigurable* because it makes it possible to change security policies on a per packet basis at runtime. That is, it assures a fine grained adaptability to possible changes in security requirements. STaR is *transparent*, because the application can still rely on the communication interface already in use. PLASA's modularity allows for loading/unloading PLASA modules to match security requirements, add new features, or extend existing ones. This clearly separates the implementation of the application from PLASA components. PLASA characteristics allow for reusing application components in scenarios where security becomes relevant. Also, PLASA is *dynamically reconfigurable* because it can switch from a security procedure to another while the application is running.

2

The IEEE 802.15.4 security sublayer

IEEE 802.15.4 is a standard addressing low-rate wireless personal area networks with a focus on enabling low power devices, personal area networks, and wireless sensor networks (WSNs). The standard is characterized by maintaining a high level of simplicity, allowing for low cost and low power implementations [7]. IEEE 802.15.4 is adopted in a wide range of application scenarios, ranging from structural monitoring to health care, from military surveillance to industrial automation. Most of these applications require forms of secure communication. For this reason, IEEE 802.15.4 specification includes a number of security provisions and options that constitute the *security sublayer* [7]. The security sublayer provides link-level security services by guaranteeing confidentiality and/or authenticity and replay detection on a per-frame basis. Specifically, it provides two *security parameters*, the *security level*, which specifies one (out of eight) possible security service, and the *key identifier mode*, which specifies one (out of four) possible way to store and lookup cryptographic keys.

Security and performance of IEEE 802.15.4 have been thoroughly analyzed. For instance, a performance analysis of IEEE 802.15.4 without considering security has been performed in quite a few papers including [8, 9, 10]. In addition, a security analysis of IEEE 802.15.4 security sublayer, its services, vulnerabilities, and related countermeasures, has been presented in [11, 12, 13]. However, a thorough analysis of the impact that the security sublayer has on the overall IEEE 802.15.4 performance is missing. Some related works have been presented but they focus on specific aspects. For example, [11, 13, 14, 15, 16, 17, 18] deal with the cost for the sensor node of using off-the-shelf ciphers, encryption modes, and authentication algorithms in terms of energy, storage and computing overhead. Other works focus instead on the cost of key establishment, an important although collateral aspect [9, 19, 20, 21]. However, what it is really missing is an analysis providing quantitative indications regarding the impact that the security sublayer has on the overall standard performance. We believe that this analysis is crucial. Security and performance compete for the same system resources, namely memory, CPU, bandwidth and energy, which are scarce in low power, low cost sensor devices. Therefore, quantitative indications regarding resource consumption are fundamental to design and implement adequate performance-security trade-offs in IEEE 802.15.4-based applications.

In this dissertation we present a performance analysis of the IEEE 802.15.4 security sublayer. In particular we evaluate the impact that security levels and key identification modes have on network performance indices such as latency, goodput, and energy consumption. The objective of our analysis is twofold. On the one hand, we aim at evaluating how security impacts on network performance, i.e., how security services (e.g. confidentiality and/or authenticity and replay detection) and security options (e.g., the length of the message authentication code) influence performance. On the other hand, we aim at devising a cost model that allows designers and implementers to carry out, for example at pre-deployment, simulation and/or performance analysis that include security too.

IEEE 802.15.4 security sublayer provides its services to above network and application layers. Although IEEE 802.15.4 security sublayer is the natural choice for ZigBee [22], nevertheless this is not the only option. Actually, different network and/or application protocols, can be deployed on top of the IEEE 802.15.4 MAC layer [23]. For this reason, we have chosen to evaluate the performance of the IEEE 802.15.4 security sublayer in isolation, irrespective of the actual network or application protocols that will be layered on top of it, so as to give our work a wider and more general scope.

We claim that our work has the following merits. First, we show that i) securing traffic has performance costs due to the increased length of a secured frame and the additional computations required for security processing; and, ii) these costs depend on the chosen security parameters. Second, we show that the highest cost has to be paid when we switch from unsecured to secured traffic. However, when traffic is secured via hardware-based cryptography, the chosen security service has little, or even negligible, impact on performance. Conversely, when traffic is secured via software-based cryptography, the performance penalty strongly depends on the chosen security level. Third, we propose a simple yet effective analytical model that we also use to extend an Ns2-based simulator of the IEEE 802.15.4 MAC protocol. The model and the extended simulator have been experimentally validated by means of real measurements on an open-source implementation of the IEEE 802.15.4 for TinyOS on Tmote Sky motes [2, 24]. Finally, the availability of an implementation of the standard has allowed us to evaluate the memory overhead related to the security sublayer. It turns out that, while the code implementing the sublayer has limited memory occupancy, the internal data structures may constitute a constraint to the system scalability.

The closest work to ours is [11]. However, in this work Chen *et al.* present a performance analysis that is only based on simulations and lacks of any experimental validation. In addition, they neglect the impact of the key identifier mode, and refer to a partial implementation of the security sublayer that fails to capture the memory costs and the consequent constraints on the system scalability.

2.1 Related work

Security of IEEE 802.15.4 has been largely investigated. Many works have focused on the analysis of the security services offered by the IEEE 802.15.4 security sublayer, its vulnerabilities, the possible attacks and related countermeasures. Among them, relevant examples are [12, 13, 25]. In addition to this, another branch of research has focused on the impact of security on performance. For instance, several works have investigated the cost of using off-the-shelf ciphers, encryption modes, and authentication algorithms on wireless sensor nodes in terms of energy consumption, storage and computing overhead. Relevant examples are [14, 15, 16, 17, 26, 27]. However, none of these works focuses on the performance implications of IEEE 802.15.4 security.

Xiao *et al.* and Zhu *et al.* explored first the impact of security on performance [13, 18]. However, these works greatly differ from ours for several reasons. They both investigate the cost of a software implementation of the ciphers, encryption modes, and authentication algorithms. Such an investigation only focuses on performance implications on a single node. In contrast, we refer to more efficient sensor node architectures where cryptographic transformations are applied at the hardware level by the communication device. Also, we focus on the overall wireless sensor network performance rather than on a single node. Last, but not the least, we refer to the current version of the standard (released in 2006 [7]) whereas Zhu *et al.* and Xiao *et al.* refer to the 2003 version [28]. These versions greatly differ in the security sublayer.

The closest work to ours is certainly [11]. Like us, Chen *et al.* refer to the 2006 version of the standard and evaluate the impact of the security sublayer on the overall network performance. They mainly focus on the influence of the packet size and inter-arrival time, whereas we mainly focus on the impact of the security level and the key identification mode. In addition, there are other strong differences. First of all, like [13, 18], Chen *et al.* consider an incomplete implementation of the security sublayer. Actually, their implementation is limited to the cryptographic transformations but completely neglects the data structures required by the security sublayer and, consequently, their impact on memory consumption. Therefore, they fail to capture an important factor limiting the overall scalability. As we consider a complete implementation, we are instead able to capture such a scalability issue (Section 2.3.3). Furthermore, they only consider a software implementation of AES-128 [29], the block cipher at the basis of the cryptographic transformations. More in details, they only refer to 20-byte payload frames and consider a 26 ms per-block encryption/decryption delay, a particularly large value derived in a previous work [30]. Instead, we consider several payload sizes (namely 2, 18, and 80 bytes), and use both hardware-based and software-based cryptography. Specifically, we consider hardware-based cryptography supported by the CC2420 communication device [31], and software-based cryptography based on an implementation of AES-128 taken from the TinyOS security algorithms repository [32]. From our experiments it turns out that hardware-based cryptography accounts for an approximately constant overhead of 1.4 ms. Furthermore, software-based cryptography introduces an initial computing delay of 0.74 ms for key scheduling and an additional computing delay of 1.93 ms for each encrypted/decrypted block (see Section 2.3.1). It follows that performance indicators reported by Chen *et. al.* in [11] result about one order of magnitude larger than ours in the case of software-based cryptography and two orders of magnitude larger in the case of hardware-based cryptography (see Section 2.3). Finally, Chen *et al.*'s analysis is only based on simulation without any experimental validation of the results. The only measurements account for the cost of software cryptography but they come from a previous paper [30]. In contrast, we present an analytical model, an extended simulator, and a set of experiments on real sensor nodes validating both the model and the simulation results. The work we present in this chapter extends a work presented in [33].

Using security mechanisms requires establishing the cryptographic keys to be used by the encryption algorithms. However, the IEEE 802.15.4 security sublayer does not specify any key establishment scheme and, for this reason, we will not discuss this issue any further in the rest of the chapter. Notwithstanding, it is important to notice here that, due to the limited resources and the large scale of WSNs, the key management scheme for desktop- and server-computing are generally not suitable. Therefore, key management and its performance in WSNs has become a very active research topic [34, 35]. Many key management schemes have been proposed and evaluated, that are ready to use in IEEE 802.15.4 [19, 34, 36, 37]. Relevant examples are [20, 38, 39, 40].

Finally, we would like to spend a comment on [41]. TinySec is not compliant with IEEE 802.15.4. Actually, it can be considered an alternative solution to link-level security. However, from a performance point of view, Karloff *et al.* achieve similar conclusions as ours. Namely, much of the overhead can be fully explained by the increased packet length and additional computations that security imposes.

2.2 IEEE 802.15.4: an overview

In this section we provide an overview of the IEEE 802.15.4 standard, with a focus on the CSMA/CA network multiple access protocol. The reader may refer to the standard [7] for further details.

IEEE 802.15.4 is a standard for low-rate, low-power *Personal Area Networks* (*PANs*). The standard defines two different types of device, namely *Reduced-Function Devices* (*RFDs*) and *Full-Function Devices* (*FFDs*). RFDs are intended to perform simple operations and typically feature minimal resources in terms of memory, storage and processing capabilities. In contrast, FFDs may have more resources and can fulfill network management tasks. A device may play one of the following roles: *ordinary device*, *coordinator*, or *PAN coordinator*. An RFD can only be an ordinary device, whereas an FFD can play any role. A network may have one or more coordinators but only one PAN coordinator that is selected among the coordinators. A coordinator is responsible to manage a subset of ordinary nodes by relaying messages among them. In order to communicate, ordinary nodes must associate with a coordinator. IEEE 802.15.4 supports two network topologies, namely *star*, and *peer to peer*. The former one is single-hop, whereas the latter is multi-hop. Also, the standard defines two channel access modes, namely, *beacon-enabled* and *nonbeacon-enabled*. In the beacon-enabled mode, the PAN coordinator periodically broadcasts beacon frames to synchronize channel access. In the nonbeacon-enabled mode, coordinators do not emit beacon frames and devices transmit frames without waiting for beacons. In this dissertation we focus on the beacon-enabled mode.

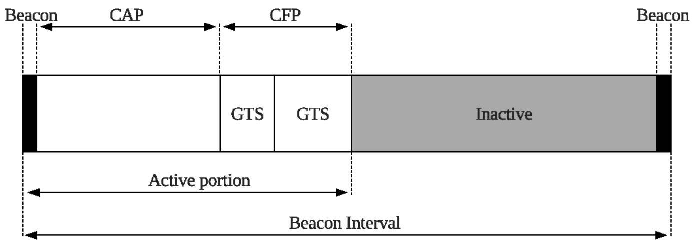

Figure 2.1: IEEE 802.15.4 superframe structure.

With reference to Figure 2.1, in the beacon-enabled mode, two consecutive beacons bound a *superframe*. A superframe is divided into *superframe slots* whose duration is 320 μs. All operations are slot-aligned. A superframe has an *active portion* and an optional *inactive portion*. The PAN coordinator can switch to low-power mode during the inactive portion.

The active portion of a superframe may be divided in two periods, the *Contention Access Period* (*CAP*) and, optionally, the *Contention Free Period* (*CFP*). The Contention Access Period starts immediately after the beacon. The Contention Free Period, if present, goes from the end of the Contention Access Period to the end of the active portion. The Contention Free Period consists in a collection of *Guaranteed Time Slots* (*GTSs*) that are allocated by the PAN coordinator to requesting devices in order to let them access the medium without contention. In this dissertation we will focus on the Contention Access Period.

In the Contention Access Period sensor nodes use the *Carrier Sense Multiple Access with Collision Avoidance* (*CSMA/CA*) protocol to access the shared communication medium and avoid collisions. The access protocol is organized in *backoff stages*. Initially, a sensor node waits for a random *backoff interval*, which is a time interval multiple of the superframe slot. At the end of this waiting, the sensor node performs two consecutive Clear Channel Assessment (CCA) operations, to ascertain that the channel is free. If the channel is found busy at least once, the sensor node starts another backoff stage with a longer backoff period (if the maximum allowed number of backoff stages is exceeded the frame is dropped). Specifically, the backoff window is doubled at each backoff stage, unless the maximum allowed value has been reached. On the contrary, if the channel results free twice, the sensor node sends the data frame and waits for the related ACK frame. Upon receiving a frame correctly, the recipient replies with an ACK without contention. If the ACK is not received within a predefined time interval, the sender retransmits the data frame (unless the maximum number of retransmissions has been exceeded).

2.2.1 IEEE 802.15.4 security sublayer

The IEEE 802.15.4 security sublayer optionally provides link-level security services to the higher layers. In general, link-level security secures the wireless link and allows applications to function at least as securely as they would do over a wired network. It follows that link-level security allows a *seamless* integration of wireless networks into existing wired networks and provides the greatest ease of deployment among currently available network cryptographic approaches [42]. Furthermore, specifically in a WSN, link-layer security supports in-network processing, passive participation and local broadcast to save traffic and reduce energy [41, 43]. The two other likely alternatives, namely end-to-end security at the application layer and end-to-end security at the transport layer, provide a high level of security, but require a complex setup of cryptographic keys, and neither guarantee seamless integration nor support in-network processing, passive participation and local broadcast. Of course, link-level security and end-to-end security mechanisms can co-exist. Security at multiple places in the protocol stack is not considered harmful and constitutes a means to respond to demand for more security with yet more sophisticated use of cryptography [42, 43].

The IEEE 802.15.4 security sublayer guarantees data confidentiality, data authenticity and replay detection on a per-frame basis. ACK frames are not secured. A frame can be secured according to *security levels*. Specifically, three different security levels are defined: the *CTR* security level provides confidentiality; the *CBC-MAC* security level provides authentication and replay detection; and, finally, the *CCM* security level provides authentication and confidentiality. In order to implement the cryptographic transformations required by the security levels, the standard uses the Advanced Encryption Standard (AES) block cipher [29]. AES has a fixed block size of 128 bits and a variable key size of 128, 192, or 256 bits. IEEE 802.15.4 uses 128-bits keys only.

IEEE 802.15.4 does not define any key establishment schemes, which are entrusted to the higher layers. In practice, the standard assumes that both senders and recipients pre-share common security settings and store the needed security material before secure communications can actually take place. However, IEEE 802.15.4 provides four *Key Identifier Modes* to identify and retrieve a cryptographic key to secure/unsecure a frame.

An unsecured frame is composed of three fields, namely a *MAC Header* (7–23 bytes), and a variable length *Payload* (0–118 bytes) and a *Frame Check Sequence* (*FCS*, 2 bytes). A secured frame contains an additional header called the *Auxiliary Security Header* (*ASH*), and, if the security level includes authentication, the *Message*

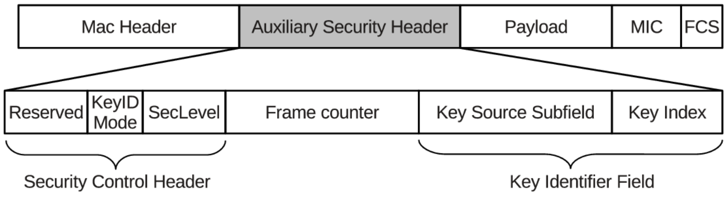

Figure 2.2: Auxiliary Security Header (ASH) structure.

Integrity Code (*MIC*). The ASH carries the information required for security processing and frame securing and unsecuring. In a secured frame, the ASH is placed next to the standard MAC header (Figure 2.2). The ASH is a 5–14 byte data structure composed of three fields: i) the *Security Control Header* (1 byte) which specifies the security level (3-bits *SecLevel* sub-field) and the *Key Identifier Mode* (2-bits *KeyIdMode* sub-field); ii) the *Frame Counter* (4 bytes) for the anti-replay service; and, finally, iii) the *Key Identifier Field* (0–9 bytes) that contains information to identify the key to unsecure a frame. The Auxiliary Security Header (ASH) is transmitted in the clear but it can be authenticated as described in the following.

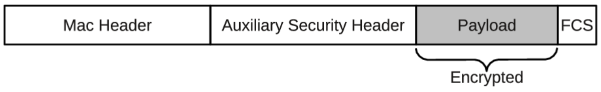

Figure 2.3: CTR security level.

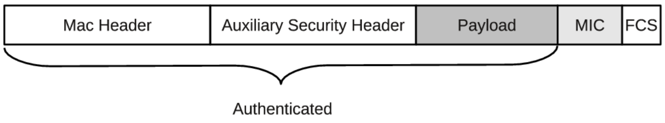

Figure 2.4: CBC-MAC security level.

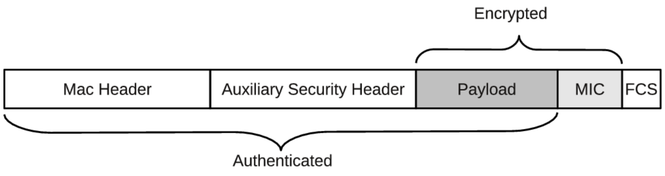

Figure 2.5: CCM security level.

Security levels are depicted in Figures 2.3, 2.4, and 2.5. The CTR security level secures a frame by encrypting its payload in the counter mode (Figure 2.3). As a rule of thumb, the CTR security level requires a block cipher encryption operation for each block to encrypt. The CBC-MAC security level secures a frame by authenticating the frame header, the ASH, and the payload (Figure 2.4). The CBC-MAC security level initially computes a 128-bit Message Integrity Code (MIC) by using the AES block cipher in the cipher-block-chaining mode. Then, the MIC is truncated and appended to the frame. The MIC can be truncated at 4, 8 or 16 bytes, so leading to three variations of CBC-MAC of increasing security, namely CBC-MAC-4, CBC-MAC-8, and CBC-MAC-16, respectively. As a rule of thumb, the CBC-MAC security level requires a block cipher encryption operation for each block to authenticate. Finally, the CCM security level secures a frame by using the AES block cipher in the counter with CBC-MAC mode (Figure 2.5). The CCM security level initially authenticates the frame header, the ASH, and the payload as in the CBC-MAC security level. Like the CBC-MAC security level, the MIC can be truncated at 4, 8, or 16 bytes so producing three variations of the CCM of increasing security, namely CCM-4, CCM-8, and CCM-16, respectively. Finally, CCM security level encrypts the resulting MIC and the payload in the counter mode. The CCM security level requires one block cipher encryption operation for each block of encrypted or authenticated fields (i.e. frame header, ASH and MIC) and two encryption operations for the payload, that is both authenticated and encrypted.

Security mode	**Data confidentiality**	**Data authenticity**	**Replay detection**	**MIC size (bytes)**
CTR	ON	OFF	ON	-
CBC-MAC-4	OFF	ON	ON	4
CBC-MAC-8	OFF	ON	ON	8
CBC-MAC-16	OFF	ON	ON	16
CCM-4	ON	ON	ON	4
CCM-8	ON	ON	ON	8
CCM-16	ON	ON	ON	16

Table 2.1: Security levels.

Table 2.1 gives an overview of the available security levels. For each security level, the table specifies the security services it provides (i.e. “Confidentiality,” “Authentication,” and “Replay detection”). If a security level introduces a MIC, column “MIC size” specifies the corresponding length in bytes.

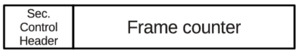

Figure 2.6: Format of ASH in KeyIdMode0.

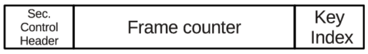

Figure 2.7: Format of ASH in KeyIdMode1.

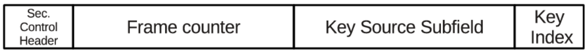

Figure 2.8: Format of ASH in KeyIdMode2.

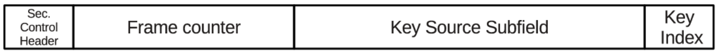

Figure 2.9: Format of ASH in KeyIdMode3.

Figures 2.6, 2.7. 2.8 and 2.9 show the format of the ASH depending on the key identifier mode. In the case of Key Identifier Mode 0 (KeyIdMode0), the ASH does not include any Key Identifier Field and security operations rely on a pre-shared static default key (Figure 2.6). In the case of Key Identifier Mode 1 (Figure 2.7), the Key Identifier Field contains the Key Index sub-field only (1 byte). In the case of Key Identifier Modes 2 and 3 (KeyIdMode2 and KeyIdMode3), the Key Identifier Field contains both the Key Index and Key Source subfields. The Key Source Subfield is four bytes in the KeyIdMode2 (Figure 2.8) and eight bytes in the KeyIdMode3 (Figure 2.9).

KeyIdMode	**ASH size (bytes)**
0	5
1	6
2	10
3	14

Table 2.2: Size of the ASH as a function of the KeyIdMode.

Table 2.2 reports the size of the Auxiliary Security Header (ASH) as a function of the key identifier mode.

2.2.2 Security operations

The standard specifies a number of *security operations*, namely the *security procedures* and *sub-procedures*. A thorough and detailed description of these operations is beyond the scope of this dissertation (the interested reader may directly refer to the standard [7]). However, in this section, we give a very concise description of the operations in order to convey the intuition of the computations they carry out and the computing overhead they imply. In particular, we highlight that security operations involve not only *cryptographic transformations* but also *management operations*, such as frame parsing and data structures lookups.

The standard considers two main security procedures, the *outgoing frame security procedure*, performed on the sending side upon frame transmission, and the *incoming frame security procedure*, performed on the receiving side upon frame reception. These procedures exploit two main data structures, the *Key Table* and the *Device Table*. The Key Table stores the cryptographic keys used by the node as well as information about the usage of these keys. Typically, the Key Table is accessed using the pair (Key Source, Key Index) as search key to retrieve the cryptographic key identified by such a pair, the list of nodes using such a key, and the types of frames (beacon, data, command) to be protected by means of such a key. The Device Table records the devices with which the node is communicating. Typically, the Device Table is accessed using the device identifier as search key to retrieve the last value of the frame counter received from that device.

The outgoing frame security procedure receives the unsecured frame, the security level, the key identifier mode, the Key Source and the Key Index as input parameters, and secures such a frame as specified by the security level, using the key identified by the pair (Key Source, Key Index) according to the key identifier mode. If the procedure succeeds, the resulting secured frame is returned for transmission. Notice that securing the frame consists in applying to the unsecured frame the cryptographic functions specified by the security level.

The incoming frame security procedure receives the secured frame and, initially, parses it and determines the values of the security level, the key identifier mode, the Key Source and the Key Index as specified in the Auxiliary Security Header. Then, the procedure unsecures the frame, as specified by the security level, using the key identified by the pair (Key Source, Key Index) according to the key identifier mode. If the procedure succeeds, the resulting unsecured frame is returned for reception. Notice that unsecuring a frame also requires checking whether the received frame is a replay or not. The procedure accomplishes this check by accessing the Device Table specifying the sending node identifier as search key, retrieving the corresponding frame counter field value, and ascertaining that this value is smaller than that contained in the secured frame.

2.2.3 The CONET open implementation of IEEE 802.15.4

We have implemented a complete and fully operational version of the standard security sublayer within an open-source implementation of IEEE 802.15.4 maintained by the TinyOS IEEE 802.15.4 Working Group [3]. The whole standard, including the security sublayer, has been implemented [24] in the nesC language for the TinyOS operating system on the Tmote Sky platform equipped with the CC2420 chipset. The security sublayer implementation can be downloaded from [2]. To the best of our knowledge, this is the first available free implementation of IEEE 802.15.4 including security services. All the experimental evaluations reported in this dissertation have been carried out on this implementation.

2.3 Evaluation

In the presence of security, the network experiences performance degradation due to two sources of overhead, namely the *communication overhead* and the *processing overhead*. The *communication overhead* is due to the extra bits that are transmitted due to security, namely, the ASH and the MIC field (if present). The processing overhead is due to the extra processing due to security procedures including parsing the ASH, looking up into tables as required by the standard procedures, and applying the cryptographic algorithms to secure/unsecure frames.

In order to quantify the impact of communication and processing overhead, we consider the following performance indices:

- *Latency* (τ), defined as the interval of time between the instant at which the source node starts the frame transmission and the instant at which the same node receives the corresponding ACK.
- *Goodput* (G), defined as the amount of useful information bits correctly received by the PAN coordinator per unit of time.
- *Per-packet energy consumption* (ϵ), defined as the total energy consumed by each sensor node divided by the number of data frames correctly delivered to the PAN coordinator.

In the goodput definition we consider only the payload and not the whole frame in order to underline the impact of the security overhead on transmission of the useful information carried by a MAC frame. The size of the payload field is always the same, irrespectively of the security level used. As a consequence, goodput decreases when security increases. This effect will be quantified in the next sections.

2.3.1 Analysis

In this section we evaluate analytically the impact of security services on the performance indices defined above. To this end, we consider a very simple network consisting of only two nodes, the PAN coordinator and a sensor node. In this setting, the sensor node always succeeds in accessing the wireless medium at the first attempt. This allows us to better understand the impact of security on performance.

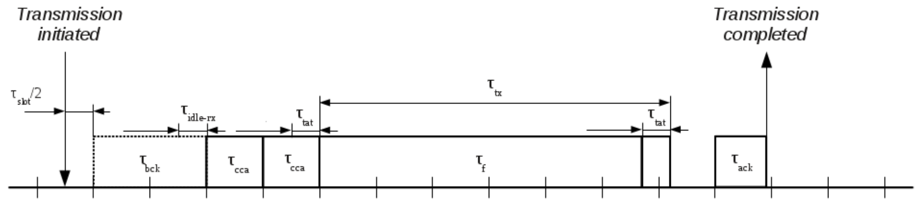

Figure 2.10: Latency timeline without security.

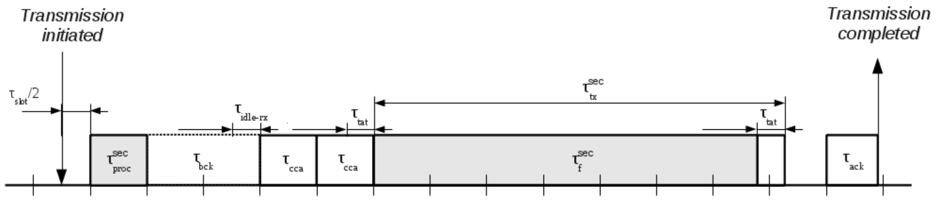

Figure 2.11: Latency timeline with security.

Latency and goodput

In order to model the impact of security on latency, we first define latency in the absence of security and, then, we add the effects of security. The average latency experienced by a frame consists of a number of components corresponding to the different steps of the CSMA/CA algorithm (see Section 2.2). As shown in Figure 2.10, assuming that the sensor node starts in the idle state, latency can be computed as:

$$\tau = \frac{\tau_{\mathrm{slot}}}{2} + \tau_{\mathrm{bck}} + 2 \cdot \tau_{\mathrm{cca}} + \tau_{\mathrm{tx}} + \tau_{\mathrm{ack}} \tag{2.1}$$

where

$$\tau_{\mathrm{tx}} = \left\lceil \frac{\tau_{\mathrm{f}} + \tau_{\mathrm{tat}}}{\tau_{\mathrm{slot}}} \right\rceil \cdot \tau_{\mathrm{slot}} \tag{2.2}$$

In Equation 2.1, $\frac{\tau_{\mathrm{slot}}}{2}$ accounts for an average delay deriving from the fact that operations are aligned to a backoff slot, whose duration is equal to τ_{slot}; τ_{bck} accounts for the random backoff time, which includes $\tau_{\mathrm{idle-rx}}$, the time necessary to switch the radio from the idle state to the receiving state; $2 \cdot \tau_{\mathrm{cca}}$ accounts for the time necessary to perform two consecutive Clear Channel Assessment operations; τ_{tx} accounts for the total time required to actually transmit a frame; and, finally, τ_{ack} is the time to receive the corresponding ACK frame. In its turn, τ_{tx} is equal to a whole number of backoff slots that contain the time interval $\tau_{\mathrm{f}} + \tau_{\mathrm{tat}}$ (see Equation 2.2), namely the *frame transmission time* τ_{f} to actually transmit a frame, and the *turnaround time* τ_{tat} to switch the radio from transmission mode to reception (and thus become able to receive the ACK frame). The *turnaround time* τ_{tat} to switch the radio from receive mode to transmission mode is part of the second τ_{cca} time interval.

Security brings in two latency contributions: the *security processing time* $\tau^{\mathrm{sec}}_{\mathrm{proc}}$, which accounts for the security processing overhead, and the *security communication time* $\tau^{\mathrm{sec}}_{\mathrm{comm}}$, which accounts for the security communication overhead. The security processing time $\tau^{\mathrm{sec}}_{\mathrm{proc}}$ accounts for the time required by security operations. The security communication time $\tau^{\mathrm{sec}}_{\mathrm{comm}}$ accounts for the time necessary to transmit the additional fields brought about by security, namely the ASH and the MIC field (when present). The communication time $\tau^{\mathrm{sec}}_{\mathrm{comm}}$ has to be added to the frame transmission time τ_{f}. With reference to Figure 2.11, it follows that Equation 2.1 becomes:

$$\tau^{\mathrm{sec}} = \tau^{\mathrm{sec}}_{\mathrm{proc}} + \frac{\tau_{\mathrm{slot}}}{2} + \tau_{\mathrm{bck}} + 2 \cdot \tau_{\mathrm{cca}} + \tau^{\mathrm{sec}}_{\mathrm{tx}} + \tau_{\mathrm{ack}} \tag{2.3}$$

where

$$\tau^{\mathrm{sec}}_{\mathrm{tx}} = \left\lceil \frac{\tau_{\mathrm{f}} + \tau^{\mathrm{sec}}_{\mathrm{comm}} + \tau_{\mathrm{tat}}}{\tau_{\mathrm{slot}}} \right\rceil \cdot \tau_{\mathrm{slot}} \tag{2.4}$$

Once we have derived analytical formulas without and with security, we can easily calculate the goodput G experienced in both cases. Assuming that the sensor node has always a frame ready for transmission, the pattern shown in Figures 2.10 and 2.11 repeats for each following frame transmission. Hence:

$$\mathrm{G} = \frac{\mathrm{P}}{\tau} \tag{2.5}$$

and

$$\mathrm{G} = \frac{\mathrm{P}}{\tau^{\mathrm{sec}}} \tag{2.6}$$

Per-packet energy consumption

Since we are assuming a network scenario with only two nodes and an ideal communication channel, the PAN coordinator receives all transmitted frames correctly. In addition, the transmission pattern for all frames is the same as the one shown in Figures 2.10 and 2.11. Hence, in order to derive the per-packet energy consumption we can refer to a single frame transmission. Specifically, we sum the energy expenditures in every time interval contributing to latency (see Equation 2.3). The energy ϵ consumed in an interval is the product of the power w consumed in τ and time interval τ itself, i.e. $\epsilon = \mathrm{w} \cdot \tau$. Power consumption can be derived from the device datasheet.

In order to evaluate the per-packet energy consumption, we observe from Section 2.2.2 that the processing overhead $\tau^{\mathrm{sec}}_{\mathrm{proc}}$ can be split into two components, namely the *management overhead*, $\tau^{\rm sec}_{\rm mgmt}$, that accounts for frame parsing and tables lookup, and the *encryption overhead*, $\tau^{\rm sec}_{\rm crypto}$, that accounts for applying cryptographic algorithms to frames. The former component is implemented in software on the sensor node microcontroller. The latter component can be implemented both in software on the sensor node microcontroller or in hardware on the radio chipset, provided this device offers hardware support to cryptography. The CC2420 radio chipset available on Tmote Sky sensor nodes provides such a support [31].

Whether cryptography is hardware-based (hw-based) or software-based (sw-based) may have a strong impact on performance for two reasons. Hardware-based encryption is faster than software-based encryption. On the other hand, hardware-based encryption is performed on the communication device that, generally, has larger power consumption than the microcontroller. In the rest of this dissertation we will evaluate performance in both cases.

Furthermore, whether cryptography is hw-based or sw-based also influences the granularity at which we are able to evaluate parameter $\tau^{\rm sec}_{\rm crypto}$. The AES algorithm consists of a key scheduling algorithm and an encryption (decryption) algorithm. Key scheduling is performed just once, before encryption (decryption) starts, whereas the encryption (decryption) algorithm is performed on each plaintext (ciphertext) block. In the sw-based cryptography case, by properly instrumenting implementation, it is possible to separate the key scheduling overhead ($\tau^{\rm sec}_{\rm key\ sw}$) from the per-block encryption (decryption) algorithm overhead ($\tau^{\rm sec}_{\rm block\ sw}$). In contrast, in the hw-based cryptography case this is not possible. It follows that the encryption processing overhead $\tau^{\rm sec}_{\rm crypto}$ will be expressed in terms of a single parameter $\tau^{\rm sec}_{\rm crypto\ hw}$ in the hw-based cryptography. In contrast, the encryption processing overhead $\tau^{\rm sec}_{\rm crypto\ sw}$ in sw-based cryptography will be expressed in terms of two parameters, $\tau^{\rm sec}_{\rm key\ sw}$ and $\tau^{\rm sec}_{\rm block\ sw}$.

Evaluation of parameters

Table 2.3 shows the parameters values for calculating Equation 2.3, assuming that the communication chipset is CC2420 [31] and the microcontroller is MSP430 [4]. The values of absorbed current referring to MSP430 and CC2420 are taken from the respective datasheets [4, 31]. The only exception is the value of the absorbed current during $\tau^{\rm sec}_{\rm crypto\ hw}$ that has been taken from [17]. The absorbed current during $\tau_{\rm idle-rx}$ has been obtained by averaging the current absorbed in the idle state and the current absorbed in the receiving state. The current absorbed during turnaround time $\tau_{\rm tat}$ has been estimated analogously (i.e., the mean value between the current absorbed in the receiving state and the current absorbed in the transmitting state). Please note that the approach we used to evaluate these currents is the same used by the Ns2 simulator to evaluate energy consumption [36, 44].

The duration of all delay components shown in Table 2.3 are derived from the standard, except for the values of $\tau^{\rm sec}_{\rm mgmt}$, $\tau^{\rm sec}_{\rm crypto\ hw}$, $\tau^{\rm sec}_{\rm key\ sw}$, and $\tau^{\rm sec}_{\rm block\ sw}$, that have been evaluated experimentally. Specifically, to measure these delays, we used two

Device	**Parmeter**	**Duration (μs)**	**Current (mA)**	**Power consumption (mW)**	**Energy consumption (μJ)**
MSP430 v = 1.8 V	Security management overhead ($\tau^{\mathrm{sec}}_{\mathrm{mgmt}}$)	260	0.6	1.08	0.28
	SW-based key scheduling overhead ($\tau^{\mathrm{sec}}_{\mathrm{key\ sw}}$)	740	0.6	1.08	0.80
	SW-based per-block cryptography overhead ($\tau^{\mathrm{sec}}_{\mathrm{block\ sw}}$)	1630	0.6	1.08	1.76
CC2420 v = 1.8 V	Total HW-based encryption overhead ($\tau^{\mathrm{sec}}_{\mathrm{crypto\ hw}}$)	1393	21.27 [17]	38.14	53.13
	Average backoff period (τ_{bck})	1120	0.427	0.77	0.86
	Slot duration (τ_{slot})	320	0.427	0.77	0.25
	Idle-rx switching ($\tau_{\mathrm{idle-rx}}$)	192	10.067	18.12	3.48
	Turnaround time (τ_{tat})	192	18.55	33.39	6.41
	Clear Channel Assessment (τ_{cca})	320	19.7	35.46	11.35
	Reception of ACK frame (τ_{ack})	352	19.7	35.46	12.48

Table 2.3: Parameters.

timers and properly instrumented our implementation of the standard (see Section 2.2.3). For the sw-based cryptography case, we used the software implementation of AES-128 algorithm that is available in the TinyOS repository [32]. In all cases, we fixed KeyIdMode3 and considered three different payload sizes, i.e., 2, 18, and 80 bytes. We measured the parameters for all possible combinations of security levels and payload sizes. For each measurement, we run an experiment consisting in sending 100 frames and taking the average. Each experiment was repeated 10 times, in order to assure a better accuracy and measure the standard deviation.

It is worthwhile to notice that time $\tau^{\mathrm{sec}}_{\mathrm{mgmt}}$ (260.61 $\pm$ 0.53 μs) accounts for the management overhead due to frame parsing and table lookups. This overhead is equal for both sw-based and hw-based cryptography and is independent of the frame size and the security level. Furthermore, in the case of hw-based cryptography, we found that, in practice, $\tau^{\mathrm{sec}}_{\mathrm{crypto\ hw}}$ (1393 μs), is influenced by neither the security level nor the payload size. In principle, $\tau^{\mathrm{sec}}_{\mathrm{crypto\ hw}}$ would depend on these parameters, which determine the actual number of blocks to be encrypted and/or authenticated. However, hw-based cryptography is so fast that its overhead is masked by the time necessary for registers setup and device strobing. Finally, in sw-based cryptography, the key scheduling overhead $\tau^{\mathrm{sec}}_{\mathrm{key\ sw}}$ and the per-block encryption overhead $\tau^{\mathrm{sec}}_{\mathrm{block\ sw}}$ are not negligible and account to 740 μs and 1630 μs, respectively. It follows that, in contrast to hw-based cryptography, $\tau^{\mathrm{sec}}_{\mathrm{crypto\ sw}}$ now greatly depends on both the payload size and the security level.

	CTR	**CBC-MAC-4 or CCM-4**	**CBC-MAC-8 or CCM-8**	**CBC-MAC-16 or CCM-16**
KeyIdMode0	5	9	13	21
KeyIdMode1	6	10	14	22
KeyIdMode2	10	14	18	26
KeyIdMode3	14	18	22	30

Table 2.4: Frame expansion due to security. Values are in bytes.

Table 2.4 shows the frame expansion (in bytes) as a function of the security level and the key identifier mode. Such an expansion is due to the ASH and the MIC, if present. The size of the former depends on the KeyIdMode (see Section 2.2.1) whereas the size of the latter depends on the security level (see Section 2.2.1).

Analytical results

In this section we show the trends of latency, goodput and energy consumption as functions of the security level. In this analysis, we consider the KeyIdMode3 that, for each security level, causes the largest ASH, therefore the largest frame expansion and thus represents the worst case from the communication viewpoint. We evaluate the trends in the case of both hw-based and sw-based cryptography for three different values of the payload, namely 2 bytes, which features a *small payload*; 18 bytes, which features a *realistic payload*; and, finally, 80 bytes, which features the *largest payload* when the MIC and ASH have the largest size.

Figures 2.12, 2.13, and 2.14 show the trend of latency, goodput and energy consumption with the security levels for different payload sizes in KeyIdMode3, when using hw-based cryptography. As it turns out, the main performance penalty occurs when we move from unsecured (NO-SEC) to secured traffic. However, a variation of the security level causes little, almost negligible, variations in the security cost. Consider latency for example. Switching from NO-SEC to CTR, causes latency to increase by the 57% in the case of 2-bytes payload, 49% in the case of 18-bytes payload, and 35% in the case of 80-bytes payload. However, switching from CTR to CCM-16 causes just a latency increase of 12%, 11%, and 8%, respectively. As to goodput, switching from NO-SEC to CTR causes a decrement of 36% in the case of 2-bytes payload,

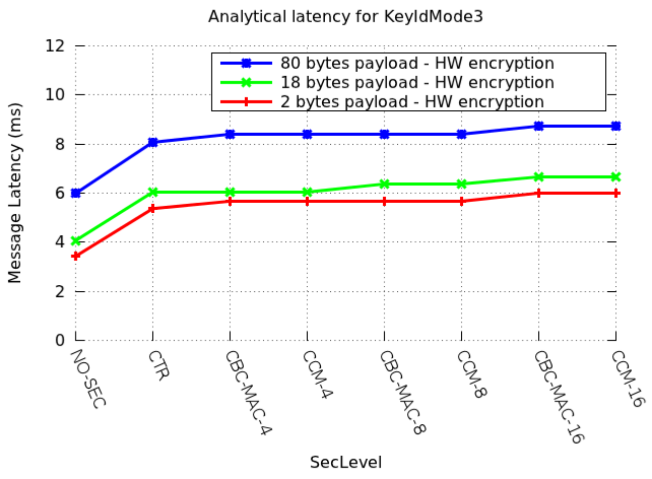

Figure 2.12: Latency for HW-based cryptography with different security levels.

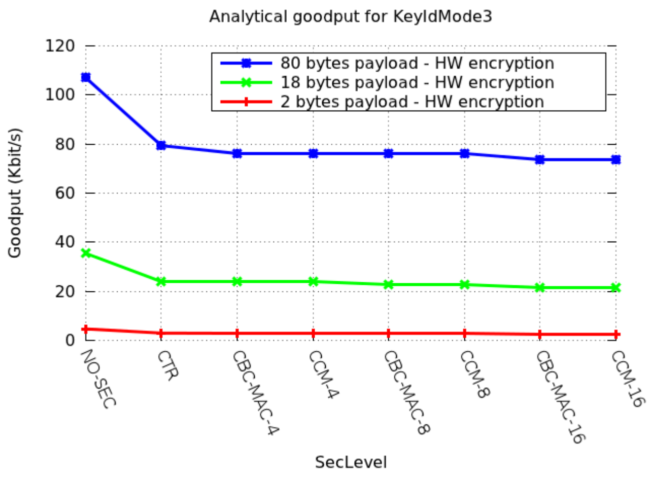

Figure 2.13: Goodput for HW-based cryptography with different security levels.

33% in the case of 18-bytes payload, and 26% in that of 80-bytes payload. However, switching from CTR to CCM-16 causes a further decrement of just 11% in the case of 2-bytes payload, 10% in the case of 18-bytes payload, and 7% in that of 80-bytes payload. Finally, as to energy consumption, switching from NO-SEC to CTR causes an increment of 89% in the case of 2-bytes payload, 71% in the case of 18-bytes payload, and 45% in that of 80-bytes payload. However, switching from CTR to CCM-16 causes a further increment of just 15% in the case of 2-bytes payload, 13% in the case of 18-bytes payload, and 5% in that of 80-bytes payload.

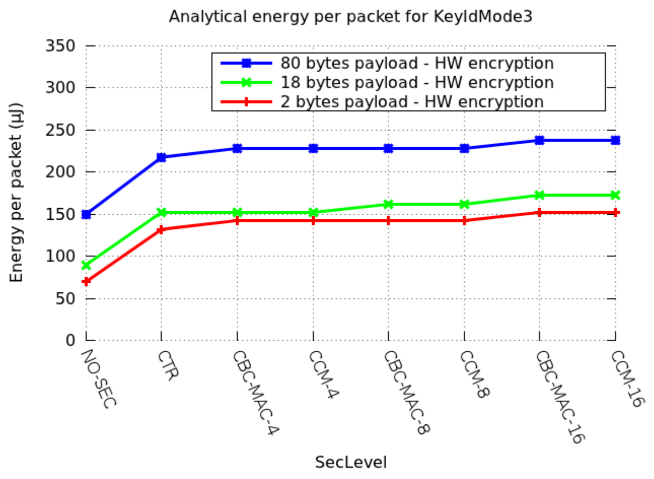

Figure 2.14: Energy for HW-based cryptography with different security levels.

It is interesting to observe that, in some cases, a change in the security level that causes a frame size increment does not reflect in a latency increase. For instance, consider the 80-bytes payload curve. Switching from CCM-4 (CBC-MAC-4) to CCM-8 (CBC-MAC-8) does not cause any latency change even though the latter implies transmitting 4 bytes more than the former. This is because the increase in the transmission time due to frame size increment is hidden by the backoff alignment, as expressed by Equation 2.4. Similar considerations hold for goodput and energy consumption.

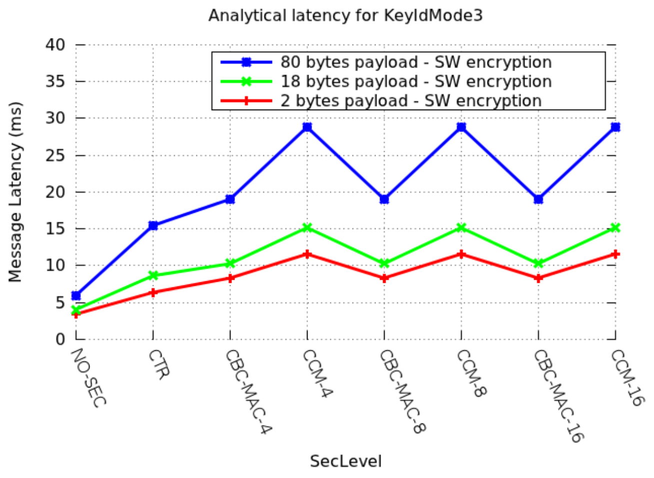

Figure 2.15: Latency for SW-based cryptography with different security levels.

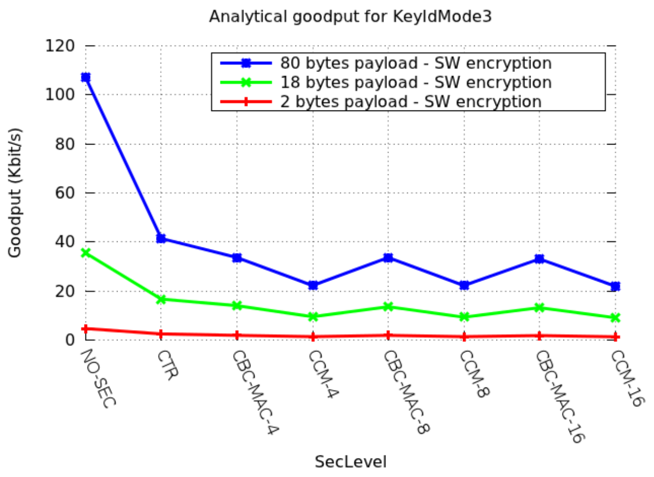

Figure 2.16: Goodput for SW-based cryptography with different security levels.

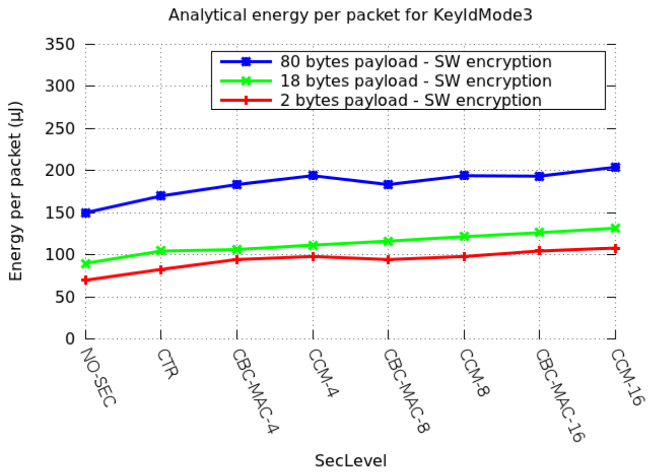

Figure 2.17: Energy for SW-based cryptography with different security levels.

Figures 2.15, 2.16, and 2.17 show the trend of latency, goodput and energy consumption with the security levels for different payload sizes in KeyIdMode3, when using sw-based cryptography. Similarly to the previous case (i.e. hw-based cryptography), a performance penalty occurs when we move from unsecured (NO-SEC) to secured traffic. However, in contrast to the previous case, variations in the security level (or payload size) cause considerable variations in the security cost. Actually, as discussed in Section 2.3.1, the security level determines the number of block encryption/decryption operations whose delays, in the case of sw-based cryptography, are not negligible.

For example, switching from NO-SEC to CTR, causes latency to increase considerably by the 86% in the case of 2-bytes payload, 112% in the case of 18-bytes payload, and 158% in the case of 80-bytes payload. However, switching from CTR to CBC-MAC causes latency to increase by about 30% in the case of 2-bytes payload, about 23% in the case of 18-byes payload and, finally, about 24% in the case of 80-bytes payload. Switching from CTR to CCM causes latency to increase by about 80% in the case of 2-bytes payload, about 79% in the case of 18-byes payload and, finally, about 87% in the case of 80-bytes payload.

Goodput has a dual behavior. Switching from NO-SEC to CTR causes a goodput decrement of 46% in the case of 2-bytes payload, 52% in the case of 18-bytes payload, and 61% in that of 80-bytes payload. Goodput further decreases upon switching to CBC-MAC and CCM.

As to per-packet energy consumption, switching from NO-SEC to CTR causes an increment of 18% in the case of 2-bytes payload, 16% in the case of 18-bytes payload, and 13% in that of 80-bytes payload. Furthermore, switching from CTR to CCM-16 causes a further increment of 15% in the case of 2-bytes payload, 13% in the case of 18-bytes payload, and, finally, 9% in the case of 80-bytes payload.

It turns out that the per-packet energy consumption is the only metric that improves upon moving from hw-based to sw-based cryptography. For instance, if we consider the CCM-16 security level, latency increases by 44% in the case of 2-bytes payload, 137% in the case of 18-bytes payload, and 235% in the case of 80-bytes payload. Consistently, goodput decreases by 50%, 58%, and 70%, respectively. In contrast, per-packet energy increases by 29%, 24%, and 14%, respectively. The reason is that, while performing cryptographic operations, MSP430 absorbs much less power than CC2420. Actually, from Table 2.3 it turns out that both devices operate at 1.08 V but MSP430 absorbs 0.6 mA, whereas CC2420 absorbs 21.19 mA, i.e. a current, and thus a power that is about 35 times larger than the former. As a consequence, even though sw-based cryptography is slower than hw-based cryptography, the overall energy consumed by the former is smaller than that consumed by the latter.

Experimental validation of the analytical model

The analytical model has been validated through experimental measurements on a real testbed. The experimental testbed consisted of Tmote Sky sensor nodes [4], equipped with an MSP430 microcontroller, 10 KB of RAM, 48 KB of ROM and, finally, a CC2420 radio transceiver. CC2420 is compliant with the IEEE 802.15.4 physical layer and supports a 250 Kbit/s bit rate over an unlicensed 2.4 GHz ISM band [31]. As to system software, sensor nodes run the TinyOS 2.x operating system (available from [45]) and the CONET open-source implementation of IEEE 802.15.4 (see Section 2.2.3). To validate the analytical results derived in previous section, we considered only two sensor nodes, KeyIdMode3 and a payload size equal to 18 bytes.

SecLevel	Experimental (HW-based cryptography)	Analytical (HW-based cryptography)	Experimental (SW-based cryptography)	Analytical (SW-based cryptography)
NO-SEC	4.28 (± 0.14)	4.06	4.28 (± 0.14)	4.06
CTR	6.35 (± 0.13)	6.04	9.08 (± 0.16)	8.64
CBC-MAC-4	6.57 (± 0.18)	6.04	10.83 (± 0.16)	10.27
CCM-4	6.51 (± 0.17)	6.04	16.51 (± 0.16)	15.16
CBC-MAC-8	6.62 (± 0.13)	6.36	11.25 (± 0.14)	10.59
CCM-8	6.79 (± 0.22)	6.36	16.62 (± 0.16)	15.48
CBC-MAC-16	6.95 (± 0.22)	6.68	11.43 (± 0.16)	10.91
CCM-16	6.99 (± 0.20)	6.68	16.75 (± 0.17)	15.80

Table 2.5: Experimental vs analytical latency. Values are in ms.

SecLevel	Experimental (HW-based cryptography)	Analytical (HW-based cryptography)	Experimental (SW-based cryptography)	Analytical (SW-based cryptography)
NO-SEC	33.62 (± 1.1)	35.43	33.62 (± 1.1)	35.43
CTR	22.69 (± 0.47)	23.85	15.86 (± 0.26)	16.66
CBC-MAC-4	21.90 (± 0.59)	23.85	13.30 (± 0.20)	14.02
CCM-4	22.12 (± 0.58)	23.85	8.72 (± 0.13)	9.50
CBC-MAC-8	21.77 (± 0.43)	22.65	12.80 (± 0.15)	13.59
CCM-8	21.20 (± 0.69)	22.65	8.67 (± 0.14)	9.30
CBC-MAC-16	20.71 (± 0.63)	21.57	12.60 (± 0.18)	13.19
CCM-16	20.60 (± 0.58)	21.57	8.60 (± 0.09)	9.11

Table 2.6: Experimental vs analytical goodput. Values are in Kbit/s.

Table 2.5 and Table 2.6 show the analytical and experimental values of latency and goodput, for different security levels, when using hw-based and sw-based cryptography, respectively. The experimental measurements are fully consistent with the analytical results. Furthermore, they completely confirm the trend we have already observed in Section 2.3.1. As far as hw-based cryptography, a significant variation in performance occurs when we proceed from unsecured to secured frames. However, the security level has little, if not negligible, influence on performance. When using sw-based cryptography, the performance loss is greater than in the case of hw-based cryptography and strongly depends on the number of block encryption operations and thus, ultimately, on the payload size and the security level.

2.3.2 Simulation analysis

In the previous analysis, we have considered a network composed of two nodes. This allows us to understand the impact of security when there is no contention between sensor nodes. In this section we consider a more complex but more realistic network composed by more nodes.

We consider a star, beacon-enabled PAN composed of a coordinator and a variable number of ordinary sensor nodes that are placed in a circle around the sink node, 10 m far from it. Upon receiving a beacon frame, an ordinary node attempts, until it succeeds, to transmit a frame to the coordinator. The Beacon Interval is 983.04 ms (BO = 6 and SO = 6).

In order to evaluate the impact of security on performance, we simulated such a network by means of the Ns2 simulation tool [46]. The basic IEEE 802.15.4 simulator has been extended to take into account $\tau_{\mathrm{proc}}^{\mathrm{sec}}$ and $\tau_{\mathrm{comm}}^{\mathrm{sec}}$. The former was modeled as a pure delay. The latter has been implemented by fictitiously enlarging the payload by a quantity specified in Table 2.4 for each relevant pair (security level, KeyIdMode). In simulations, we only considered KeyIdMode3. We have set the transmission range to 15 m and the carrier sensing range to 30 m as in [47]. In addition, we have considered an 18-bytes payload corresponding to a total unsecured frame size of 33 bytes. We derived simulation results for both hw-based and sw-based cryptography.

For each simulation, we have performed 10 independent repetitions, each consisting of 1000 Beacon Intervals each. The presented results are averaged over the ten replications with a 95% confidence level. For each repetition, we discarded the initial transient period during which nodes associate to the PAN coordinator before starting generating data packets.

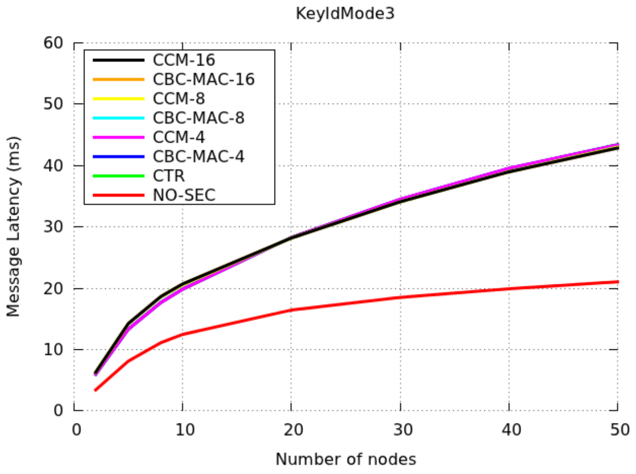

Figure 2.18: Simulated latency for HW-based cryptography.

As to hw-based cryptography, Figures 2.18, 2.19 and 2.20 show the simulation trend of latency, goodput, and per-packet energy consumption with the number of

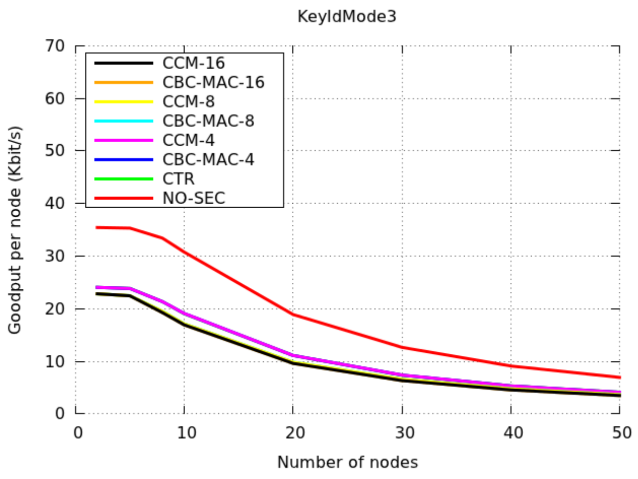

Figure 2.19: Simulated goodput for HW-based cryptography.

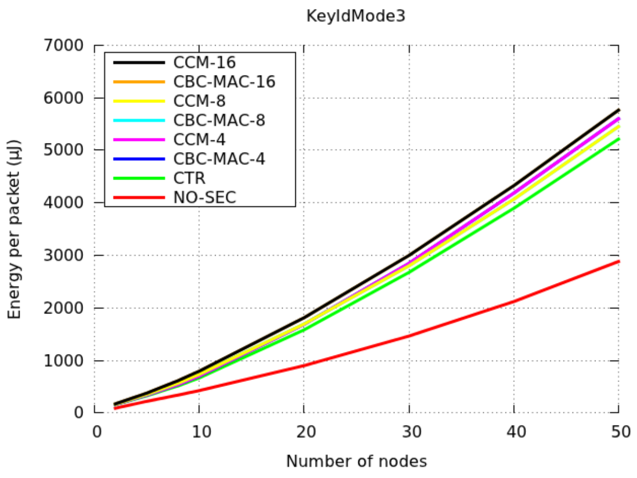

Figure 2.20: Simulated energy consumption for HW-based cryptography.

nodes for each security level. Confidence intervals are so small that they cannot be graphically appreciated.

As above, we validated our simulation results through experimental measurements. Table 2.7 compares the simulation and experimental results (and the corresponding confidence intervals), for latency and goodput with two and ten nodes. As it turns out, simulation and experimental results agree with each other.

At first glance, we may observe that, in accordance with the previous analysis, for any given number of nodes, switching from unsecured to secured traffic causes a neat performance loss due to the security processing and communication overhead. How-

		Latency		Goodput	
	SecLevel	**Experimental latency (ms)**	**Simulative latency (ms)**	**Experimental goodput (%)**	**Simulative goodput (%)**
2 nodes	NO-SEC	4.28 (± 0.14)	3.42 (± 0.01)	33.62 (± 1.1)	35.44 (± 0)
	CTR	6.35 (± 0.13)	5.98 (± 0.01)	22.69 (± 0.47)	24.07(± 0)
	CBC-MAC-4	6.57 (± 0.18)	5.98 (± 0.01)	21.90 (± 0.59)	24.07(± 0)
	CCM-4	6.51 (± 0.17)	5.98 (± 0.01)	22.12 (± 0.58)	24.07 (± 0)
	CBC-MAC-8	6.62 (±0.13)	6.30 (± 0.01)	21.77 (± 0.43)	22.84 (± 0)
	CCM-8	6.79 (± 0.22)	6.30 (± 0.01)	21.20 (± 0.69)	22.84 (± 0)
	CBC-MAC-16	6.95 (± 0.22)	6.30 (± 0.01)	20.71 (± 0.63)	22.84 (± 0)
	CCM-16	6.99 (± 0.20)	6.30 (± 0.01)	20.60 (± 0.58)	22.84 (± 0)
10 nodes	NO-SEC	13.12 (± 0.14)	12.47 (± 0.03)	29.34 (± 0.96)	30.76 (± 0.05)
	CTR	19.14 (± 0.58)	19.84 (± 0.03)	21.40 (± 0.45)	19.03 (± 0.02)
	CBC-MAC-4	19.71 (± 0.61)	19.85 (± 0.03)	21.44 (± 0.57)	19.09 (± 0.04)
	CCM-4	19.51 (± 0.54)	19.85 (± 0.03)	20.66 (± 0.54)	19.09 (± 0.04)
	CBC-MAC-8	19.43 (± 0.43)	20.54 (± 0.04)	20.55 (± 0.41)	17.12 (± 0.02)
	CCM-8	20.10 (± 0.37)	20.54 (± 0.04)	20.40 (± 0.66)	17.12 (± 0.02)
	CBC-MAC-16	19.41 (± 0.50)	20.69 (± 0.05)	20.05 (± 0.61)	16.92 (± 0.04)
	CCM-16	20.33 (± 0.53)	20.69 (± 0.05)	18.58 (± 0.52)	16.92 (± 0.04)

Table 2.7: Simulative vs. experimental latency and goodput with HW-based encryption.

ever, the specific security level has little, or even no influence on such a loss. Going into more details, let us consider the trend of latency (Figure 2.18). For each security level, latency increases with the number of nodes. This depends on the fact that, when the number of nodes increases, it is more likely that a node attempting to transmit has to wait for the free medium. Also, the probability of collisions increases and, hence, some frames have to be retransmitted. However, it turns out that the latency in the case of secured traffic grows with the number of nodes more quickly than the latency in the case of unsecured traffic. Actually, curves tend to diverge. This depends on the additional delays deriving from the security processing and communication overhead that every transmitting node brings in. Due to this additional delay, *ceteris paribus*, in the case of secured traffic, node experiences a latency longer than in the case of unsecured traffic. Goodput has a dual trend (Figure 2.19), with respect to latency.

Similar considerations also apply to the energy consumption per delivered packet (Figure 2.20). The increasing trend is more remarkable than latency because not only the total energy consumption increases with the number of sensor nodes, but the percentage of delivered frames decreases, as emphasized by the goodput decrease in Figure 2.19.

As to sw-based cryptography, Figures 2.21, 2.22 and 2.23 show the trend of latency, goodput and per-packet energy consumption with the number of nodes for each

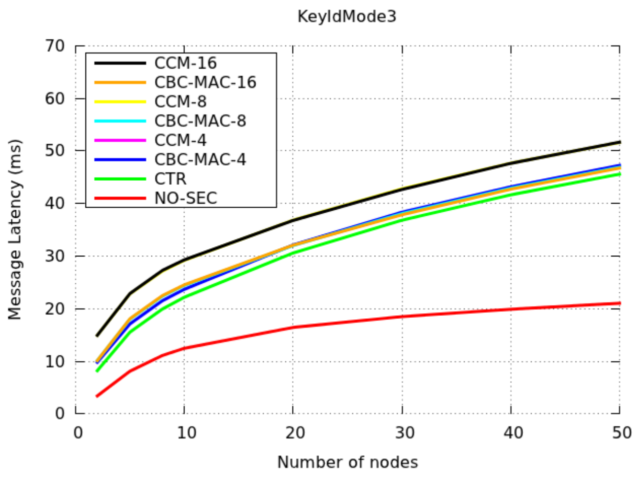

Figure 2.21: Simulated latency for SW-based cryptography.

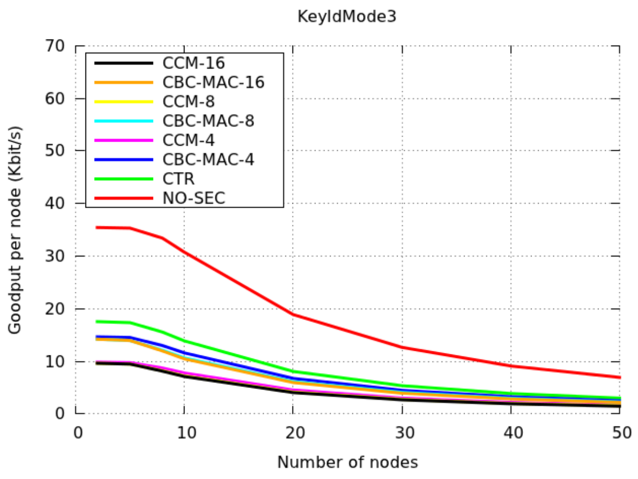

Figure 2.22: Simulated goodput for SW-based cryptography.

security level. As above, confidence intervals are so small that they cannot be graphically appreciated.

As expected, Figures 2.21, 2.22 and 2.23 show that switching from unsecured to secured traffic causes a performance loss. Furthermore, figures also show that payload size and security level have influence on such a loss, due to the number of block encryption operations that are required. However, Figure 2.23 shows that per-packet energy consumption constitutes an exception and its trend is very similar to the hardware-based cryptography (Figure 2.20). This is because, with respect to

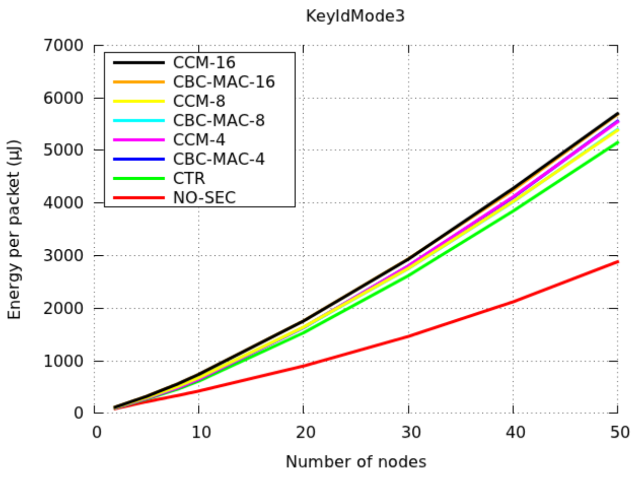

Figure 2.23: Simulated energy consumption for SW-based cryptography.

hw-based cryptography, sw-based cryptography increases the encryption processing overhead $\tau_{\mathrm{crypto}}^{\mathrm{sec}}$ but, at the same time, requires a lower power consumption.

As in the previous case, we validated our simulation results through experimental measurements. Again, we observed a general agreement between simulation and experimental results. We omit them for the sake of space.

2.3.3 Experimental evaluation of memory overhead

In this section we evaluate, through an experimental analysis carried out with the testbed described in Section 2.3.1, the memory overhead introduced by the IEEE 802.15.4 security sublayer.

Figures 2.24 and 2.25 show the ROM footprint breakdown on both the PAN coordinator and a regular sensor node. With hw-based cryptography (Figure 2.24), the amount of memory required by the security sublayer executable is the 11.58% of the overall memory available on the PAN coordinator, and the 12.96% on a regular sensor node. In both cases, most of the memory occupancy is due to the IEEE 802.15.4 implementation (i.e. the original communication stack). Note also that the 19.46% (15.44%) of memory on the PAN coordinator (regular node) remains available for other uses (e.g., applications). With sw-based cryptography (Figure 2.25), the amount of memory required by the security sublayer executable is the 17.36% of the overall memory available on the PAN coordinator, and the 18.76% on a regular sensor node. In both cases, most of the memory occupancy is due to the IEEE 802.15.4 implementation and software-based implementation of the encryption algorithm. Note also that 13.68% (9.66%) of memory on the PAN coordinator (regular node) remains available for other uses (e.g., applications).

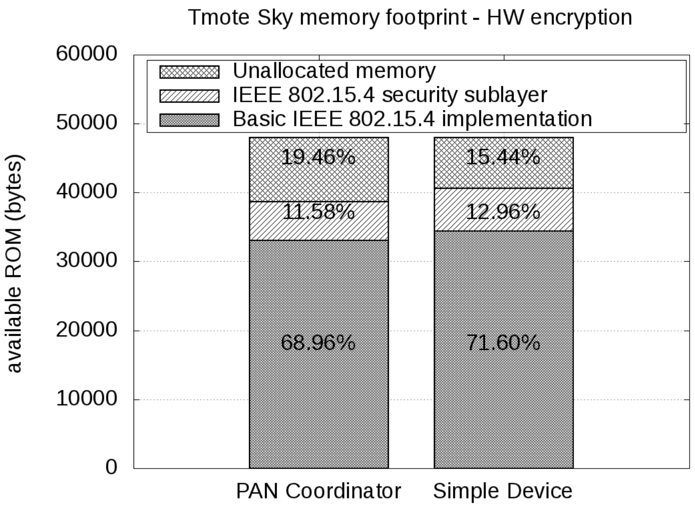

Figure 2.24: ROM memory overhead with HW-based cryptography.

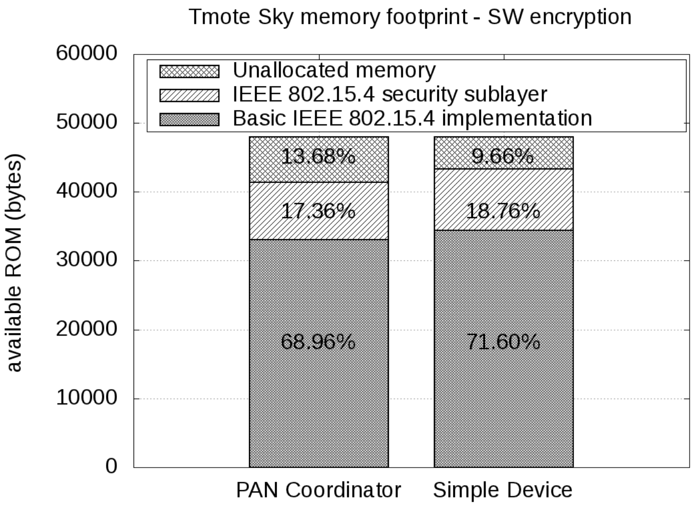

Figure 2.25: ROM memory overhead with SW-based cryptography.

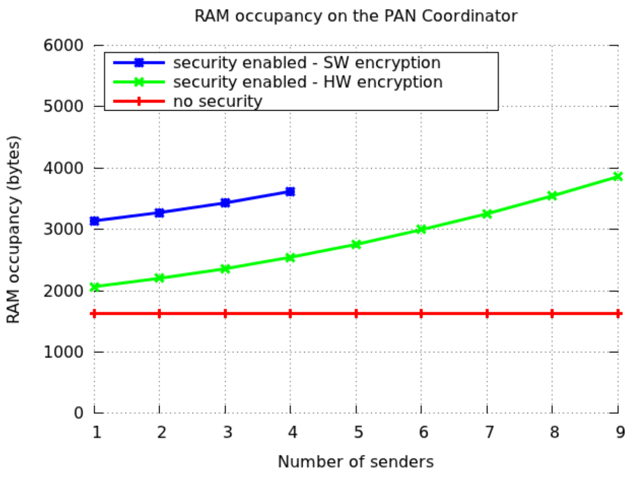

Figure 2.26: RAM memory overhead.

However, the space necessary to allocate executable is not the only storage cost that we have to pay in order to use the security sublayer. As discussed in Section 2.2.2, the security sublayer requires data structures, e.g., the Device Table and the Key Table, that are allocated in RAM and whose size grows with the number of nodes and keys. Figure 2.26 shows the trend of RAM occupancy when the number of nodes grows. In our implementation, 9 sender nodes require about 3858 bytes of RAM with hardware encryption. Beyond this threshold, we experimentally observe that motes hang or behave erratically.

In the case of sw-based cryptography, we have to allocate in RAM also the data structures of the AES encryption algorithm, which account for about 1 Kbytes. It follows that the threshold is crossed with a smaller number of nodes, namely four.

With TinyOS/msp430-gcc, there is no limit, but the physical capacity, to the amount of memory that a software component may use. However, it is not recommended to fill up the entire RAM with the component variables, because TinyOS needs space for the stack. There is no straightforward way to calculate the amount of memory TinyOS needs. However, as a rule of thumb, it is better to leave at least 500 byte or 1 KB empty, otherwise mote might hang or do erratic things because of a stack overflow.

Of course, we cannot exclude that a more efficient implementation than ours may get a greater threshold. However, regardless the actual value of the threshold, the important point to capture here is that in memory scarce devices, the amount of memory necessary for security data structures may constitute a limit to the system scalability.

3

Selective Denial of Service in IEEE 802.15.4 networks

The IEEE 802.15.4 standard [7] allows devices to access the medium not only in contention mode but also in a contention-free way, in order to support *Quality of Service* (*QoS*). In contention-free mode, devices access the medium according to the *Guaranteed Time Slot* (*GTS*) mechanism, which allows devices to access the medium without contention [48].

In a PAN, the medium access temporization is divided into consecutive *superframes*. Each superframe is divided into a *Contention Access Period* (*CAP*) and a *Contention Free Period* (*CFP*). In the CFP, nodes access the medium during pre-assigned slots, namely *GTS slots* (hereafter *Slot*s). At the beginning of a superframe, network devices require Slots to the *PAN coordinator*, which allocates available Slots to requiring nodes, and returns them the *GTS List* specifying its allocation decision.

Unfortunately, the PAN coordinator transmits the GTS List in the clear. Therefore, an adversary simply equipped with a radio receiver/transmitter can easily eavesdrop the allocation decision, select a Slot, and jam it. We call this kind of *Denial of Service* (*DoS*) attack the *GTS-based selective jamming attack*.

With respect to a classical wide-band jamming where the adversary jams the whole channel, a GTS-based selective jamming attack defines a different trade-off between attack severity and attack detectability. A wide-band jamming attack has the highest severity but is the simplest to detect. In contrast, a selective jamming is much more difficult to detect because the adversary limits its exposure to a Slot. However, it may cause severe QoS degradation to specific traffic segments.

Sokullu *et al.* have first identified this type of attack and illustrated two possible incarnations, namely the *random attack* and the *intelligent attack* [5]. In the random attack, the adversary selects the Slot to jam at random. In contrast, in the intelligent attack, the adversary exploits the knowledge of the allocation decision to select the longest Slot. Furthermore, Sokullu *et al.* have evaluated that an intelligent attacker can achieve a corruption strength of 50.48% [6], which means that only half of the available bandwidth would be actually available for communication during the CFP. It turns out that an intelligent attack makes it possible to compromise the QoS of the whole network. Notwithstanding the efficiency and the severity of this type of attack, no countermeasures have been devised before the work we presented in [49].

In this dissertation we fill this gap by a twofold contribution. First of all, we complete the range of possible GTS-based selective jamming incarnations with the *sniper attack*. In this attack, the adversary selects a victim node and then, exploiting the knowledge of the allocation decision, jams the Slot allocated to that node. It follows that this attack may compromise the QoS of a specific node, or even thwart its communication capability altogether, with the minimum chances of being detected.

Secondly, we present a standard-compliant countermeasure against the GTS-based selective jamming attack, i.e. *Selective Jamming Resistant GTS* (*SJRG*). When SJRG is active, an attacker can do no better than a random attack (of course this attack, as well as the wide-band jamming, is inevitable). SJRG is based on two basic mechanisms: i) protection of secrecy and integrity of beacon and GTS request frames by means of encryption; and ii) protection from network traffic analysis by means of intra-Slot randomization. While several solutions can be devised, the challenge is to devise one that is compliant with the IEEE 802.15.4 standard. We took up the challenge and conceived the aforementioned mechanisms in such a way that SJRG is fully compliant to IEEE 802.15.4.

3.1 Wireless Denial of Service: the jamming attack

Denial of Service (*DoS*) attacks are a threat which is vital to take into account while securing wireless communications. DoS attacks can be referred to any event that diminishes or eliminates a network's capability to perform its expected functions [50]. In other words, DoS attacks target availability by preventing communication between network devices or by preventing a single device from sending traffic [51].

Jamming consists in corrupting messages transmitted by legitimate users, by interfering in the network's operational frequencies. Jamming is one of the most common DoS attacks, and is considered a severe issue in wireless communications [52, 53, 54, 55, 56]. In the rest of this section, we consider i) techniques typically adopted to detect and defeat jamming; ii) the effectiveness of jamming against wireless communication; and iii) how different kinds of jamming attacks are classified.

Typical techniques to detect jamming attacks consist in analyzing: i) the received signal strength indicator; ii) the average time required to sense an idle channel; and iii) the packet delivery ratio [56].

On the other hand, the most adopted defense against the jamming attack relies on spread-spectrum communication among network devices [51, 57, 58]. This countermeasure requires the attacker either to follow the adopted hopping sequence, or to interfere with a wide section of the band. Another solution relies on legitimate network nodes, which collaboratively identify the jammed region in order to route traffic around it [50]. This requires to adopt a proper routing protocol, such as the TinyOS

Destination-Sequenced Distance-Vector Routing [59], which determines high-quality links according to associated link quality estimators.

It has been proven that link layer jamming particularly affects wireless networks performance. In [60], the authors discuss a selective jamming attack, according to which the adversary disturbs the transmission of specific and particularly important kinds of packets. Also, they show the effectiveness of selective jamming on the TCP protocol and its performance. Finally, they propose some methods based on cryptographic primitives, aimed at mitigating its effects.

In [61], the authors define an *intelligent jammer* from the energy consumption point of view. Since in a WSN it is reasonable to assume that the attacker is a sensor node, energy is an issue even for the attacker. Since the intelligent jammer knows MAC protocol specifications, she can preserve energy by attacking at a specific time. On the contrary, a blind jammer wastes energy emitting a continuous signal without any knowledge of the medium access criteria.

An adversary can perform different kinds of jamming, and different classifications of such an attack have been proposed so far. In [56], Xu *et al.* focus on Wireless Sensor Networks, and classify jamming attacks as *constant*, *deceptive*, *random*, and *reactive*. A constant jammer aims at corrupting all network packets by continuously transmitting random signals. Note that such an "always-on" jamming strategy is easier to detect, since it is based on the continuous presence of an high interference level [52, 55, 56]. The deceptive attack consists in injecting a constant stream of bytes into the network, making it look as legitimate traffic. Instead, a random jammer performs the attack by alternating a sleep phase with a jamming phase, thus reducing energy consumption. Finally, a reactive jammer, such as the sniper attacker, performs jamming only when she detects a transmission by other nodes. Of course, reactive jamming results to be much more difficult to be detected, since it is likely to be confused with regular collisions.

O'Flynn considers IEEE 802.15.4 networks, and refers to a different classification of jamming attacks [62], i.e. i) a wide-band jamming of all available channels; ii) a specific jamming performed upon detecting transmissions of IEEE 802.15.4 messages; and iii) a much more precise jamming against specific messages or network nodes. The sniper attacker performs a jamming of the third kind, and she is particularly dangerous since she perfectly knows when her target transmits. So, she does not need to read the first several bytes of IEEE 802.15.4 MAC headers.

Wood *et al.* take into account IEEE 802.15.4 networks, and present DEEJAM, a protocol which provides a number of defenses against energy-efficient jamming attacks in IEEE 802.15.4 networks [63]. The authors classify jamming attacks into four categories, namely *interrupt jamming*, *activity jamming*, *scan jamming*, and *pulse jamming*. DEEJAM aims at hiding messages from a jammer node, evading its search, and reducing the impact of messages that are corrupted anyway. In particular, it makes use of channel hopping, or uses a pseudo-random sequence as Start of Frame Delimiter at the physical-layer. As a result, the DEEJAM protocol maintains a packet delivery ratio of up to 88%. However, several Wireless Sensor Networks applications are supposed to rely on approved standard, as IEEE 802.15.4. Therefore, DEEJAM can not be considered an official protocol and is not likely to be widely adopted.

3.2 IEEE 802.15.4 MAC overview

In this section, we briefly summarize the main features of IEEE 802.15.4, with reference to the *Guaranteed Time Slots* (*GTS*) mechanism and available security services. Table 3.1 provides the reader with a list of acronyms we use throughout this chapter.

Acronym	**Term**
AES	Advanced Encryption Standard
ASH	Auxiliary Security Header
CAP	Contention Access Period
CBC-MAC	Cipher Block Chaining Message Authentication Code
CCM	Counter with CBC-MAC (mode of operation)
CFP	Contention-Free Period
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTR	Counter Mode
FCFS	First Come First Served
FCS	Frame Check Sequence
FFD	Full-Function Device
GTS	Guaranteed Time Slot
MAC	Medium Access Control
MFR	MAC Footer
MHR	MAC Header
MIC	Message Integrity Code
PAN	Personal Area Network
RFD	Reduced-Function Device

Table 3.1: List of acronyms.

As described by the IEEE 802.15.4 standard [7], *Personal Area Networks* (*PANs*) can be composed of two types of devices, namely *Full-Function Devices* (*FFDs*) and *Reduced-Function Devices* (*RFDs*). FFDs can communicate with both FFDs and RFDs, while RFDs can communicate only with other FFDs. Also, one specific FFD is elected as the *PAN coordinator*, and is responsible for network management. Two possible topologies are admitted: *Star* and *Peer-to-peer*. In the Star topology each RFD communicates directly with the PAN coordinator, whereas in the Peer-to-peer topology each device can communicate with any other FFD in its range. In the rest of this chapter, we consider the Star topology, and simply refer to an RFD as a *node*.

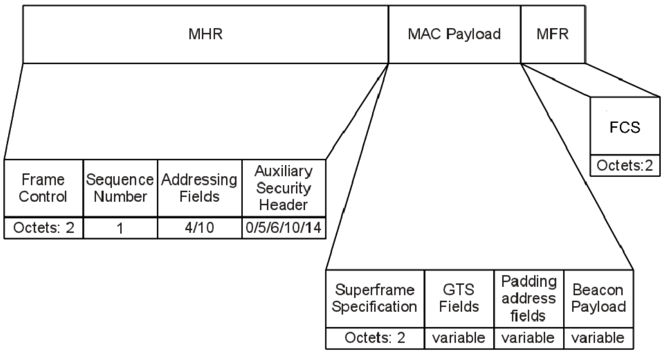

Figure 3.1: Beacon frame format.

As specified by the standard, PANs can work in two possible ways, namely *nonbeacon-enabled* or *beacon-enabled* mode. In particular, in nonbeacon-enabled mode, frames are transmitted according to an *unslotted* Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) algorithm. In case the medium is sensed idle, the transmission starts immediately. Otherwise, a device delays the transmission for an exponential random backoff time. In beacon-enabled networks, the *PAN coordinator* periodically broadcasts *beacon frames* in order to synchronize devices. Each device transmits frames according to a *slotted* CSMA/CA algorithm. The structure of a *beacon* frame is shown in Figure 3.1.

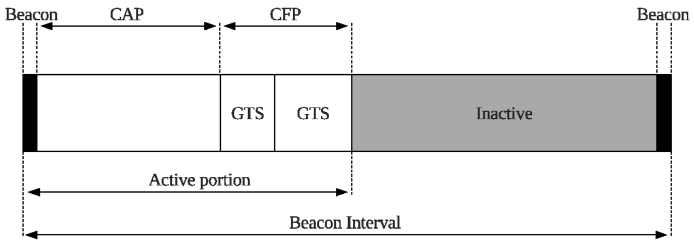

Figure 3.2: IEEE 802.15.4 superframe structure.

In beacon-enabled mode, the PAN coordinator bounds the medium access temporization as a sequence of *superframes*. As shown in Figure 3.2, each superframe is bounded by two consecutive beacon frames, periodically transmitted by the PAN coordinator. A superframe can have an *active portion* and an *inactive portion*. During the inactive portion, the PAN coordinator may switch to low-power mode to save energy. The active portion consists of 16 equally sized *superframe slots*. The active portion includes a *Contention Access Period* (*CAP*) and an optional *Contention Free Period* (*CFP*). During the CAP, nodes access the medium on a contention basis, according to a slotted CSMA/CA algorithm. On the other hand, devices may ask the PAN coordinator for dedicated portions of the CFP, *Slots* in our parlance, in order to access the medium without contention. This mechanism is known as *Guaranteed Time Slot* (*GTS*) (see Section 3.2.1) , and is particularly useful for applications with QoS constraints and requirements, such as low latency or particular bandwidth requirements.

3.2.1 Guaranteed Time Slot (GTS)

CFP allows nodes to access the medium during pre-assigned superframe slots, according to the *GTS* mechanism of IEEE 802.15.4 [7]. Slots are allocated within the CFP by the PAN coordinator, and are composed by one or more superframe slots.

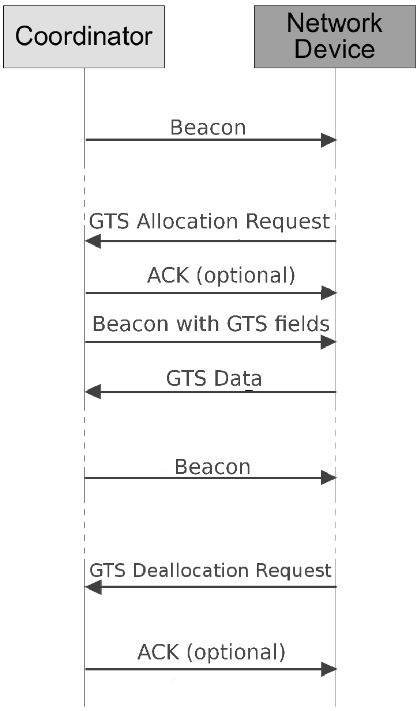

Figure 3.3: GTS allocation and deallocation requests.

Figure 3.3 shows the sequence of operations which take place during a GTS allocation/deallocation process. A node can ask the PAN coordinator for one Slot, specifying the amount of superframe slots needed and the traffic direction, i.e. transmission or reception. If the request is accepted, one Slot is reserved to that specific user for its own transmissions/receptions. When the assigned Slot is no more needed, a node requests the PAN coordinator to deallocate it. Both allocations and deallocations are requested by MAC command frames, transmitted anytime nodes successfully access the medium during the CAP.

On the other hand, the PAN coordinator manages a pool of seven Slots in a *First Come First Served* (*FCFS*) fashion. Until there are still superframe slots available in the CFP, the PAN coordinator provides requesting nodes with one Slot each. Each Slot size is the amount of superframe slots specified in the associated GTS request. The information carried within the GTS fields of each beacon frame specifies i) whether the node's request has been accepted or not; and ii) when the node is supposed to exclusively access the medium during the CFP. By doing so, a number of users can access the medium without colliding with each other.

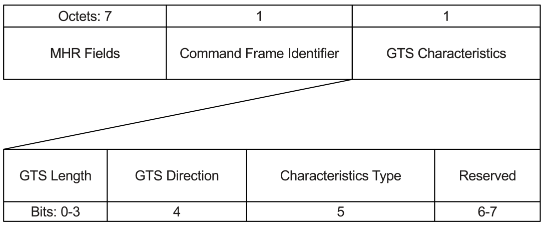

Figure 3.4: GTS request command format.

Both allocation and deallocation requests are issued by means of a MAC Command frame, namely *GTS Request Command*, whose structure is shown in Figure 3.4. The *GTS Characteristics* field is the most significant field in GTS allocation requests, and is composed of the following subfields: i) the *GTS Length*, which specifies the number of superframe slots requested for the Slot; ii) the *GTS Direction*, which specifies the direction of the data frame transmission; and, finally, iii) the *Characteristics Type*, which distinguishes between GTS allocation and GTS deallocation.

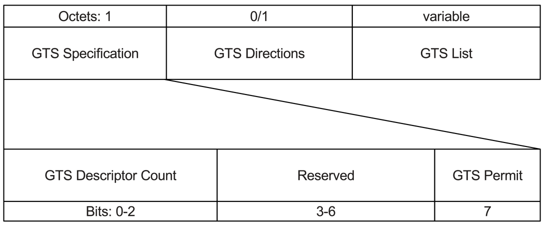

Figure 3.5: Format of the GTS fields.

Upon receiving a GTS Request Command, the PAN coordinator may send back an optional ACK. Then, the PAN coordinator decides whether to allocate a Slot, considering the GTS requests specifications and the current available capacity in the superframe. GTS requests are considered in a FCFS fashion, and assigned Slots are placed contiguously starting from the end of the CAP. Finally, GTS related information is carried in the *GTS fields* of beacon frames (see Figure 3.1).

Figure 3.5 shows the format of such GTS fields, which consist of: i) the *GTS Specification*, which specifies if GTS is allowed or not and defines the size of the GTS List field; ii) the *GTS Directions*, which identify the directions of Slots in the superframe; and, finally, iii) the *GTS List*, which includes *GTS descriptors* representing the satisfied GTS requests.

Bit: 0-15	16-19	20-23
Device Short Address	GTS Starting Slot	GTS Length

Figure 3.6: Format of the GTS descriptor.

Figure 3.6 shows the format of a GTS descriptor. Each one of them is 3 bytes long, and is composed of the following fields: i) the *Device Short Address*, which contains the short address of the device for which the GTS descriptor is intended; ii) the *GTS Starting Slot*, which contains the superframe slot at which the Slot begins; and, finally, iii) the *GTS Length*, which contains the number of contiguous superframe slots over which GTS is active.

Every GTS requesting node continues to track beacon frames for at most *aGTS-DescPersistenceTime* superframes, in order to check if its request has been accepted [7]. If this is the case, the requesting node extracts the GTS Starting slot from the right GTS descriptor, thus gaining knowledge of its own transmission/reception time. Thanks to the GTS Starting slot, each satisfied node knows when it can transmit or receive frames without any contention to access the medium. Otherwise, if a GTS allocation request can not be satisfied, the unsatisfied node notifies a failure to the next upper layer. If a Slot is no longer required, it can be deallocated at any time, at the discretion of the PAN coordinator or the devices that originally issued the request.

3.2.2 Security services

IEEE 802.15.4 provides also a number of security services, and makes them available to the higher layers. The standard provides data confidentiality, data authenticity, and replay protection on a per-frame basis. If communications are secured, senders build an *Auxiliary Security Header* (*ASH*), insert it next to the standard MAC header, and secure frames before transmitting them. According to the information carried within the ASH, recipients retrieve the right cryptographic key and unsecure MAC frames.

The standard includes a security suite based on the *Advanced Encryption Standard* (*AES*) 128 bits symmetric-key cryptography [29]. Besides, three different security modes are available, i.e. encryption only (*CTR*); authentication only (*CBC-MAC*); and, finally, both encryption and authentication (*CCM*). Both CBC-MAC and CCM rely on a *Message Integrity Code* (*MIC*), whose size can be either 4, 8, or 16 bytes. By means of these security tools, *security data structures* and *security procedures* provided by the standard, it is possible to secure/unsecure MAC frames and contrast the GTS-based selective jamming attack.

3.3 GTS-based selective jamming attack

As described in Section 3.2.1, the PAN coordinator manages GTS allocation requests, and notifies the accepted ones by broadcasting unencrypted beacon frames. However, as defined by Sokullu *et al.* [5, 6], the intelligent attacker exploits the knowledge of the allocation decision in order to find out the longest Slot, and selectively jam it.

Instead, the sniper attacker we defined exploits the transmission of unsecured beacon frames in order to know which users have been granted a collision-free Slot. Then, the attacker creates collisions only during the Slot of a specific user, resulting in a *Denial of Service* (*DoS*) attack against it.

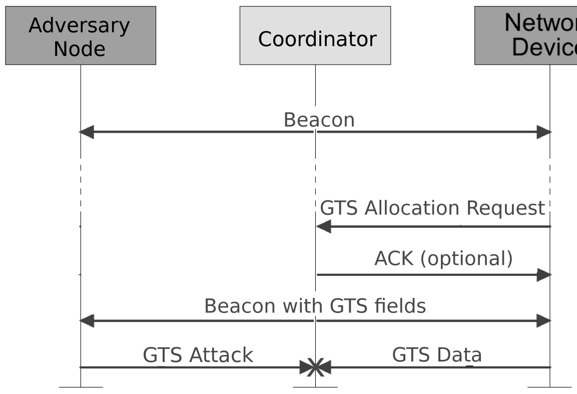

Figure 3.7: Example of GTS-based selective jamming attack.

Figure 3.7 shows an example of GTS-based selective jamming attack. By eavesdropping the medium, an adversary is able to extract the GTS List from the GTS fields of the beacon frame (see Figure 3.5). Thus, she can gain knowledge of how Slots have been scheduled within the superframe. In other words, the adversary becomes aware of which specific users are going to access the medium during the CFP of the current superframe, and during which specific Slot each one of them will access the medium.

This means it is very easy for the adversary to *selectively* interfere with transmissions, causing collisions and corruptions of data frames between the legitimate GTS clients and the PAN coordinator. The sniper attacker performs the GTS-based selective jamming attack as follows.

1. She selects her victim, that is, she picks a specific node among network devices.
2. She collects beacon frames and parses their MAC headers. By doing so, she figures whether her victim has been assigned a Slot in the CFP of the current

superframe. If this is the case, she jams the Slot assigned to her victim, selectively interfering with her transmission/reception.

3.4 Selective Jamming Resistant GTS (SJRG)

As explained in Section 3.3, an adversary can intercept broadcast beacon frames, retrieve the GTS descriptors from them, and perform the attack discussed above. The actual vulnerability in the GTS mechanism consists in the adversary having free access to the information carried within the GTS descriptors. A solution to this consists in *encrypting* and *authenticating* GTS-related information within beacon frames.

Encryption makes it possible to prevent an adversary from knowing Slots allocation. Also, authentication assures that beacon frames have been actually built by the PAN coordinator. Of course, beacons can be secured by means of the security services provided by the IEEE 802.15.4 standard. Unfortunately, the standard allows for encrypting just the beacon payload portion of the beacon MAC payload (see Figure 3.1). In contrast, the information we need to protect is carried within the GTS fields.

In this section, we describe *Selective Jamming Resistant GTS* (*SJRG*), our solution to the GTS-based selective jamming attack. SJRG is compliant with the IEEE 802.15.4 standard, and effective against both the intelligent attacker [5, 6] and the sniper attacker. Our countermeasure consists of the following steps: i) MAC frames smart encryption and authentication; and ii) random Slots allocation. The main objective of our solution is to prevent an attacker from gaining access to the information carried within the GTS Fields of beacon frames (see Section 3.2.1). SJRG manages such fields as follows.

The *GTS Specification* field contains the GTS Descriptor Count subfield, and specifies the number of GTS descriptors contained in the *GTS List*. The GTS List is a list of *GTS descriptors*, and is moved to the beacon payload portion of the beacon frame, so making it possible to encrypt and authenticate it. Finally, the *GTS Directions* subfield is moved to the beacon payload portion, so making it possible to secure it.

According to our countermeasure, the information required to successfully perform the attack is moved inside the beacon payload. As a consequence, such information can now be encrypted, so that the attacker is not able to correctly retrieve and analyze it. The PAN coordinator and the nodes which make use of this countermeasure must be able to mutually recognize each other. In order to do that, it is sufficient to rely on a proper SJRG flag in beacon frames and GTS Request commands. As to the beacon frames, we use one bit of the Reserved subfield of the GTS Specification field (see Figure 3.5). As to the GTS Request commands, we use one bit of the Reserved subfield of the GTS Characteristics field (see Figure 3.4). We believe that these two bits are the only part of SJRG which may result in a modification of the IEEE 802.15.4 standard.

It is useful to encrypt and authenticate also MAC command frames. By doing so, the adversary would not be able to analyze network traffic and recognize GTS Request Commands. If MAC command frames are encrypted, the adversary can still recognize them from their MAC header, but cannot either recognize a GTS Request Command nor distinguish between GTS allocation and deallocation requests. Authentication is needed as well, otherwise the adversary would be able to spread fake GTS Request Commands. Encryption and authentication rely on a fresh *nonce* value, i.e. a randomly generated number used to prevent replay attacks.

The standard states that the PAN coordinator manages the Slots assignment in a static, predictable way. So, if no deallocations occur, assigned Slots are not meant to change their position in the CFP, even for a considerable amount of consecutive superframes. Thus, even if the adversary cannot analyze the encrypted GTS List, she is still able to infer this information by analyzing the network traffic pattern, and observe the sequence of transmissions during the CFP. This analysis can be made pointless by unpredictably changing the position of Slots on a per superframe basis.

Unpredictably changing the position of Slots on a per superframe basis makes it possible to practically preclude an intelligent attacker or a sniper attacker from purposely causing collisions during a specific Slot. Since only the order of Slots in the CFP is changed, our solution is not in conflict with the IEEE 802.15.4 FCFS scheduling policy. In fact, GTS requests are still served in the same way by the PAN coordinator, thus SJRG does not affect the amount of accepted GTS requests.

Thanks to SJRG, the sniper attacker is not able to purposely interfere during a specific Slot. As a consequence, she can attempt a collision only on a randomly picked Slot. SJRG reduces the probability of success of the sniper attacker to x/n, where x and n are the size in superframe slots of the target Slot and the CFP, respectively. The worst case for the sniper attacker is when all Slots have been assigned and have size equal to 1 superframe slot. In such a case, since the maximum allowed GTS allocation is 7 Slots [7], the sniper attacker has a statistical success rate of $1/7$.

What follows is the sequence of actions that take place at the beginning of each superframe in the presence of SJRG.

1. Every node interested in requesting a Slot prepares a GTS Request command frame. Then, these nodes specify the number of required superframe slots, and set the SJRG flag in the Reserved subfield of the GTS Characteristics field (see Figure 3.4). Finally, they authenticate and encrypt the GTS Request command frame, and send it to the PAN coordinator.
2. The PAN coordinator verifies the authenticity of GTS Request commands, and decrypts them. Then, for each one of them, it verifies that the SJRG flag in the Reserved subfield of the GTS Characteristics field is set.
3. The PAN coordinator serves GTS requests in an FCFS fashion, according to IEEE 802.15.4 standard specifications. Once GTS requests have been served, the Slots allocation is randomly altered.
4. The PAN coordinator builds the beacon for the current superframe as follows.

   a) The GTS Directions and GTS List fields are filled, according to the output from step 3. These fields are placed into the Beacon Payload field, instead of the GTS fields.
   b) The GTS Descriptor Count subfield of the GTS Specification field is set to 0.
   c) The SJRG flag in the Reserved subfield of the GTS Specification field is set.
   d) The beacon frame is authenticated. Then, the PAN coordinator encrypts the Beacon Payload field, and broadcasts the beacon frame to network nodes.
5. Upon reception of a beacon frame, each node verifies its authenticity. Then, each node which has issued a GTS request performs the following actions.
   a) It verifies that the SJRG flag in the Reserved subfield of the GTS Specification field is set.
   b) It decrypts the Beacon Payload field, and verifies the authenticity of the beacon frame. Then, it retrieves the GTS List from the Beacon Payload, and checks if its GTS request has been accepted, i.e. if it has been granted a Slot for the current superframe.

It is worth clarifying that SJRG is not a countermeasure against the wide-band jamming attack. In fact, it is not effective in case an adversary interferes with all nodes' transmissions during the CFP, by continuously jamming all available channels. Still, we believe that, in a WSN, it is very likely that the attacker relies on sensor nodes and aims at limiting energy consumption, thus avoiding performing wide-band jamming.

3.4.1 Discussion on dictionary attack

Since SJRG requires the encryption of a small amount of bytes to protect the allocation of Slots, there might still be a chance for an adversary to find a breach in SJRG by performing a *dictionary attack*.

In order to understand what a dictionary attack consists in, consider an attacker who eavesdrops the medium, records all possible encrypted GTS Lists, and builds a dictionary. Such a dictionary allows the adversary to associate the behavior of GTS devices to the corresponding GTS List, even if it is encrypted. Each GTS List includes the Slot during which devices are supposed to transmit. If the adversary succeeds in building the dictionary, she can i) listen to a beacon frame and retrieve the encrypted GTS List of the current superframe; ii) look for the corresponding entry in the dictionary; and iii) find the exact Slot to jam in order to hit her victim. Of course, this would make SJRG useless against selective jamming.

In the following, we show how such a dictionary attack is not practically feasible. We recall that the cryptographic key used to authenticate and encrypt MAC frames has to be renewed when the Frame Counter field value is 0xffffffff, as specified by the IEEE 802.15.4 standard [7]. Also, we assume that all MAC data frames are sent by network devices to the PAN coordinator, thus the GTS Directions content remains constant over time. This is reasonable, because in many applications the PAN coordinator just collects information transmitted by sensor nodes.

According to the IEEE 802.15.4 standard, we can have up to 7 Slots per superframe. Thus, by randomly altering Slots allocation, we can have up to $7! = 5040$ possible GTS List configurations, carried within beacon payloads. Also, the IEEE 802.15.4 standard requires to encrypt each frame considering a 13 bytes nonce. Even if the nonce has a predictable structure, it includes a 32 bits long frame counter, which varies at each frame. Then, a complete dictionary consists of $5040 \cdot 2^{32}$ entries. Each entry includes one possible GTS List configuration with 7 GTS descriptors, whose size is 3 bytes each. Since we have assumed that the GTS Directions content remains constant over time, it is not necessary to include such a field in the dictionary, so each entry is 21 bytes in size.

Thus, building a complete dictionary would be practically unfeasible, since it requires more than 413 TB to be stored, which is not affordable for a wide range of adversaries. Also, if we considered a security mode which includes both encryption and authentication, we would have some extra unpredictability due to the extra bytes of the MIC. As a consequence, the dictionary would get even larger in terms of both number of entries and their size. Moreover, in order to have a complete dictionary, the adversary would have to build all the possible $5040 \cdot 2^{32}$ entries. This is impossible because the Frame Counter field value reaches 0xffffffff after 2^{32} beacons, which is less than the amount of dictionary entries. Thus, the adversary cannot complete the dictionary before key renewal takes place.

3.5 SJRG implementation

We implemented SJRG referring to the implementation of IEEE 802.15.4 for the TinyOS platform [3]. We extended the release, and implemented IEEE 802.15.4 security services and procedures. We defined modules that implement security data structures and security procedures described by the IEEE 802.15.4 standard [24], with reference to the Tmote Sky platform [4] and the CC2420 chipset [31].

As to SJRG, we extended the MAC frames parsing process, in order to properly manage the ASH in the presence of secured MAC beacon and command frames. In this section, we consider the CCM-16 security mode, which authenticates frames using a 16 bytes MIC, and then encrypts both the MIC and the payload. Nevertheless, SJRG works properly also with CCM-4 and CCM-8 security modes (see Section 3.2.2 for an overview about security modes). In case the PAN coordinator and network devices rely on SJRG, they mutually recognize each other by means of the SJRG flag carried within beacon frames and GTS Request commands (see Section 3.4).

As already discussed, it is vital also to exchange Slots positions in an unpredictable way. Otherwise, the adversary would be able to predict the new allocation scheme, and successfully perform the GTS attack. Therefore, the random reorganization of Slots within the MAC frame must rely on a *secure pseudo-random number generator* (*SPRNG*). We implemented a SPRNG based on the CC2420 chipset [31]. Specifically, we took inspiration from the ANSI X9.17 pseudo-random bit generator

[64], and considered a practical variation in order to build our own one. Specifically, our randomization function can be expressed as follows.

$$\mathrm{x_i} = \begin{cases} E_k(\mathrm{s}) & \text{if } i = 1 \\ E_k(\mathrm{x_{i-1}}) & \text{if } i \neq 1 \end{cases}$$

The input of the encryption function E is a key k and a quantity to be encrypted. The seed s must be a fresh quantity, i.e. a nonce value, and can be initialized during the PAN coordinator startup. Since the adversary does not know the key k, the output $\mathrm{x_i}$ of the above function can be considered a random variable. Note that if we assume that the key k is a secret, then the seed s can be public.

After the PAN coordinator has performed the regular Slots allocation, the order of Slots within the CFP is altered by adding all GTS starting Slots with a pseudo-random quantity generated with the SPRNG. As a result, the adversary cannot obtain information about Slots allocation by either analyzing the network traffic or observing the order of transmissions during the CFP.

3.6 SJRG evaluation

We evaluate SJRG by considering four different aspects: i) *effectiveness*, i.e. the gain in terms of delivery ratio achieved with respect to an unprotected setup; ii) *memory footprint*, i.e. the extra amount of memory required by our implementation; iii) *network performance*, i.e. the delay due to additional processing and transmissions; and, finally, iv) the additional *per packet energy consumption*.

We performed our evaluation by taking into account a real application on a realistic scenario. We considered an IEEE 802.15.4 star topology consisting of one PAN coordinator and 7 RFD nodes acting as GTS senders. Also, we considered the presence of one sniper attacker. All nodes were Tmote Sky motes, which feature a CC2420 chipset and are provided with a 48 kB ROM [4].

In our testing application, the PAN coordinator broadcasts a beacon frame, so that sender nodes can associate to the PAN. Then, all sender nodes ask for a Slot to the PAN coordinator, specifying that they support SJRG. Since the PAN coordinator provides SJRG, it broadcasts a SJRG beacon frame. Once they have received the SJRG beacon frame, sender nodes gain knowledge of their reserved Slot. From then on, each one of them transmits its data frames to the PAN coordinator during the assigned Slot. The attacker node aims at disrupting data frames transmission of a specific sender, but SJRG forces it to pick a Slot as target in a random way.

3.6.1 Effectiveness

The effectiveness of our countermeasure has been evaluated considering the probability of success of the attacker to jam communications of her target node, both in the presence and in the absence of SJRG. The presence of 7 senders represents the worst case from the attacker point of view, because the PAN coordinator always allocates 7 Slots in the CFP. Also, we fixed the size of Slots to one superframe slot each. That is, the smaller the target is, the harder it is for the attacker to hit it.

In order to estimate how many successful transmissions have been made, we observed the amount of acknowledgments received by each node that transmits during its own Slot, in the presence of the attacker node. In order to increase the accuracy of our results, we performed 10 repetitions of 100 transmissions for each experiment. The results we present here are averaged over all the different repetitions. We also report the standard deviation we derived from the independent replication method.

Our experimental evaluation considered the following two scenarios:

- *No SJRG*: the 88.4% of transmissions from the target node were corrupted by the sniper attacker, while no transmissions from other sender nodes were corrupted. The 11.6% of successful transmissions from the target node are due to imperfect clock sychronization of Tmote Sky motes. That is, because of the clock drift effect, a short frame might be transmitted before the adversary starts jamming the target.
- *SJRG*: the 13.8% of transmissions from the target node were corrupted by the sniper attacker, and the 13.7% from the other sender nodes were corrupted as well. This is due to the fact that the attacker cannot recognize her target anymore, and jams one Slot, by picking it at random. Thus, in the presence of SJRG, all sender nodes have a certain probability of being jammed. Nevertheless, the percentage of failure is affordable, and can be handled by means of retransmissions.

The amount of data frames sent by the target node and correctly received by the PAN coordinator confirms our theoretical assumption, i.e. the attacker has a statistical success rate of $1/7$. So, no Denial of Service (DoS) occurs.

3.6.2 Memory footprint

The amount of memory required by our implementation has been evaluated by comparing the TinyOS image size for Tmote Sky motes, both in the presence and in the absence of SJRG. We evaluated memory consumption considering the most complex and complete standard security configuration (i.e. KeyIdMode3 key retrieval and CCM-16 security mode), as a worst case.

As shown in Figure 3.8, the extra amount of memory is mostly due to the implementation of the IEEE 802.15.4 security sublayer. Specifically, we highlight the following five contributions: i) basic IEEE 802.15.4 implementation; ii) IEEE 802.15.4 security sublayer; iii) IEEE 802.15.4 GTS; iv) SJRG; and v) unallocated memory.

Our implementation of SJRG occupies the 1.95% of the whole Tmote Sky memory for the PAN coordinator, and the 1.08% for sender devices. We believe that such an extra memory consumption is reasonable and affordable. Finally, even in the presence of SJRG, there is still a not negligible amount of unallocated memory, both on the PAN coordinator and sender devices. This memory can be used for additional features or more complex applications.

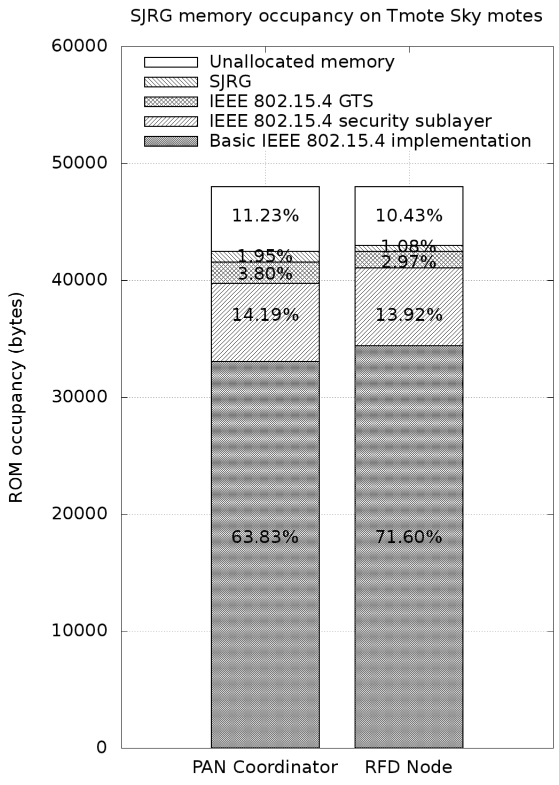

Figure 3.8: SJRG memory occupancy on Tmote Sky motes.

3.6.3 Network performance

As to network performance, SJRG is operating on top of the 2.4 GHz physical layer, with a 250 Kb/s bit rate [31]. We modeled the impact of SJRG on network performance by considering two main aspects, namely *processing overhead* and *transmission overhead* experienced by the PAN coordinator.

The processing overhead consists of two contributions: i) the hardware encryption and authentication contribution, due to the CC2420 chipset; and ii) the software contribution, due to the MSP430 microcontroller. In order to evaluate the software contribution, we considered these two components separately.

In Table 3.2, the first line reports the hardware encryption and authentication contribution. The second line shows the processing overhead due to software security procedures. The third line reports the processing overhead introduced by SJRG. Note that only the third line regards SJRG, while the first and second ones show contributions due to the IEEE 802.15.4 security sublayer. These values are averaged over all the different repetitions. We also include the standard deviation we derived from the independent replication method.

Contribution	Processing overhead (μs)	Standard deviation (μs)
HW IEEE 802.15.4 security	254.47	1.34
SW IEEE 802.15.4 security	1752.26	5.46
SJRG	125.83	2.12

Table 3.2: SJRG processing overhead contributions.

The transmission overhead has been evaluated analytically, considering a bit rate equal to 250 Kb/s [31]. We have considered the time required to transmit the additional bytes added by SJRG, according to the standard security policy. The size of the original GTS beacon packet to manage 7 Slots is 34 bytes, including the standard frame header and the *Cyclic Redundancy Check* (*CRC*). Since we consider SJRG relying on the most complete IEEE 802.15.4 security services (i.e. KeyIdMode3 key retrieval and CCM-16 security mode), the beacon frame size is increased of 31 bytes, of which 30 are due to the IEEE 802.15.4 security sublayer.

More in details: i) 14 bytes are due to the presence of the ASH; ii) 16 bytes are required to provide frame authentication; and iii) 1 byte is due to the fact that we split the GTS fields, duplicate the GTS Direction subfield, and place it into the Beacon Payload field. We keep a fake copy of the GTS Direction subfield in its original position, in order to stay compliant with the standard.

We computed the transmission time $\mathrm{d_{tx}}$ of a SJRG beacon frame as the ratio between the frame size in bits and the bit-rate:

$$\mathrm{d_{tx}} = \frac{65 \cdot 8}{0.250} = 2080\mu\mathrm{s}$$

The original beacon frame is 34 bytes in size, so it can be transmitted in

$$\mathrm{d_{tx}} = \frac{34 \cdot 8}{0.250} = 1088\mu\mathrm{s}$$

so the transmission overhead is

$$2080\mu\mathrm{s} - 1088\mu\mathrm{s} = 992\mu\mathrm{s}$$

Note that SJRG adds only one byte to the beacon payload, while the 14 bytes of the ASH and the 16 bytes of the MIC are due to the IEEE 802.15.4 security policies.

However, the user can trade off security and efficiency, thus reducing the transmission overhead due to the presence of the ASH and the MIC. IEEE 802.15.4 allows for choosing among i) different key retrieval methods, which influence the ASH size, and ii) different sizes of the MIC field, in case authentication or encryption and authentication are required. Table 3.3 provides an overview of different ASH and MIC sizes, with their associated transmission overheads. It is evident that by properly matching the ASH and MIC sizes, it is possible to reduce the transmission overhead.

Key retrieval policy	**ASH size (bytes)**	**Transmission overhead (μs)**
KeyIdMode3	14	448
KeyIdMode2	10	320
KeyIdMode1	6	192
KeyIdMode0	5	160
Security mode	**MIC size (bytes)**	**Transmission overhead (μs)**
CBC-MAC-16/CCM-16	16	512
CBC-MAC-8/CCM-8	8	256
CBC-MAC-4/CCM-4	4	128

Table 3.3: SJRG transmission overhead contributions.

Our results show that the transmission overhead introduced by SJRG is mostly due to the IEEE 802.15.4 security sublayer. Even in case all Slots are in use, SJRG adds only one byte to the beacon frame, which means 32 μs of additional transmission delay. If some Slots are not allocated, SJRG assumes that all Slots are present, as explained in Section 3.4. Of course, this increases the beacon frame size. However, beacon frames are transmitted once per superframe, which means that one beacon is broadcast every 0.983 seconds (considering BeaconOrder = 6 and SuperframeOrder = 6 [7]). Thus, we believe that the increase of beacon frames size is affordable [33, 38].

3.6.4 Energy consumption

As to energy consumption, we considered processing and transmission contributions separately. Each contribution has the form $\mathcal{E}_i = \mathcal{P}_i \cdot d_i$. We define d_i as the delay contribution due to operation i, and refer to delay values reported in Section 3.6.3. $\mathcal{P}_i = V_i \cdot I_i$ is the single power contribution, expressed as the product between voltage and current of the MSP430 microcontroller and the CC2420 chipset, responsible for processing and transmission, respectively [4]. Table 3.4 provides an overview of such contributions.

Transmission	$\mathcal{P}_{\mathbf{tx}}$ (mW)	$\mathbf{d_{tx}}$ (μs)	$\mathcal{E}_{\mathbf{tx}}$ (nJ)
	31.32	992	31069.44
Processing	$\mathcal{P}_{\mathbf{HWsec}}$ (mW)	$\mathbf{d_{HWsec}}$ (μs)	$\mathcal{E}_{\mathbf{HWsec}}$ (nJ)
	31.32	254.47	7970
	$\mathcal{P}_{\mathrm{SWsec}}$ (mW)	$\mathrm{d_{SWsec}}$ (μs)	$\mathcal{E}_{\mathrm{SWsec}}$ (nJ)
	1.08	1752.26	1892.44
	$\mathcal{P}_{\mathrm{SJRG}}$ (mW)	$\mathrm{d_{SJRG}}$ (μs)	$\mathcal{E}_{\mathrm{SJRG}}$ (nJ)
	1.08	125.83	135.90

Table 3.4: SJRG energy consumption contributions.

Considerable increases in per packet energy consumption are due to the transmission overhead of the extra bytes required by the IEEE 802.15.4 security sublayer. Also the processing overhead of standard encryption and authentication algorithms has a considerable impact on energy consumption. However, these contributions can be reduced by changing the size of the ASH and the MIC to be transmitted, as discussed in Section 3.6.3.

The actual SJRG contribution to energy consumption is the one reported in the last entry of Table 3.4, that is the energy consumed to add the SJRG field to the beacon frame after having computed the Slots order. We believe that this additional energy consumption is affordable, if compared to the contributions introduced by standard security mechanisms, including their transmission overhead.

3.6.5 On scalability

In this section, we argue that scalability of SJRG with respect to the number of users and the number of attackers is not an issue.

As to the number of users, the IEEE 802.15.4 standard admits up to 7 GTS users during the CFP, thus practically limiting the amount of GTS users to be considered in the performance analysis of SJRG. At the same time, GTS users which have been granted a Slot, and thus potential victims of the attacker, do not contend with each other to access the medium. As a consequence, although the total number of users in the system may impact the GTS mechanism, such a number does not affect SJRG.

As to the number of attackers, we argue that considering two or more attackers is not so interesting, besides being beyond the threat model we consider in this dissertation. In fact, consider the borderline case in which several attackers, independently of each other, perform a random attack, as SJRG prevents them from performing the intelligent and the sniper attack. In such a case, the greater the number of attackers, the greater the likelihood that they jam all Slots, so causing a total jamming of the collision-free portion of the channel. However, this attack would be easily detectable. As a consequence, several independent attackers would hamper each other.

4 PLASA

In the past, WSNs were mainly used for scientific purposes, where an adversary had little incentive to attack sensors [1]. In the recent years, Wireless Sensor Networks (WSNs) have been adopted in an increasing number of application scenarios. In particular, WSNs are employed in i) Cooperating Objects Systems (COSs) [65, 66] ii) Critical Infrastructures (CIs) [67, 68]. In COs, mobile physical agents share the same environment with the WSN to fulfill their tasks, either in group or in isolation. In CIs, sensor nodes collect data and transmit them to a central base station through a wireless network. In both scenarios, sensor nodes are resource constrained devices deployed in unattended, possibly hostile environments.

Given the nature of WSNs and the wireless channel vulnerabilities, it is an easy task for an adversary to eavesdrop messages and alter them, or inject fake ones. Possible consequences of a security infringement may translate into damages to things and injures to people. The more pervasive these applications become, the more pressing security guarantees get. It follows that, in all these scenarios, is vital to add security features to these applications. The main security requirements to be satisfied include secure communication, key management and secure bootstrapping.

In this chapter we present PLASA, a modular and transparent security suite to provide WSNs with secure communications, key management and secure bootstrapping. PLASA is *modular*, this makes it easily portable on different hardware, such as [69, 70] and network stacks [4, 7, 22]. Modularity allows for PLASA customization to match security requirements, add new features, or extend existing ones. PLASA characteristics allow for reusing application components in scenarios where security becomes relevant. Also, PLASA is *dynamically reconfigurable* because it can switch from a security procedure to another when the application is running.

STaR is the module of PLASA that guarantees confidentiality, integrity, and authenticity of communications by means of a set of encryption and/or authentication protocols, both hardware and software. It allows for protecting multiple traffic flows at the same time, according to different security policies. STaR is *reconfigurable* because it makes it possible to change security policies on a per packet basis at runtime. That is, it assures a fine grained adaptability to possible changes in security requirements. STaR is *transparent*, because the application can still rely on the communication interface already in use. The application does not require to be redesigned or recoded in order to exploit a certain security policy. This clearly separates the implementation of the application from the STaR component. STaR allows people that do not have a security background to secure their applications by simply selecting security policies to be applied. Besides, application developers need neither to implement complex security procedures, nor to configure unfriendly tools, such as network firewalls.

STaR integration in applications guarantees the software engineering principle of *separation of concerns* [71]. In fact, the STaR developer and the application developer can be two distinct persons that just agreed on the interface functions used to configure STaR. In other words, the application developer can adapt the system behavior to new security requirements just calling some STaR functions that have been implemented by the STaR developer, that is a security expert. Finally, STaR assures a fine grained adaptability to possible changes in security requirements and fine tuning of performance vs. resource consumption trade-offs.

PLASA features a generic architecture, and can be implemented for any hardware platform and WSNs operating system. We evaluated our PLASA implementation [72] for TinyOS [3] on Tmote Sky motes [4]. We believe the best way to evaluate PLASA efficiency is to rely on typical WSNs performance metrics and compare them both in the presence and in the absence of different security modules. Thus, we evaluated memory occupancy, communication overhead, and energy consumption both in the presence and in the absence of different PLASA modules.

4.1 Related work

Many solutions have been devised for WSNs security, including [41] for secure communication, [73, 74, 75, 76, 77, 78] for key management, and [79, 80] for secure code dissemination. However, the major part of security solutions for WSNs are well-suited to particular applications or platforms, and lack the possibility to be ported, extended or composed with others. However, only a small fraction of existing security solutions appears to be general enough to match requirements of different applications, especially when the environment changes dynamically and presents a component-based architecture [81, 82]. In [83] a transparent and reconfigurable security middleware is presented. However, we believe that a performance evaluation of a tool that introduces security in applications that present real-time constraints is fundamental. If security costs are not known in advance, security operations may cause an unaffordable performance degradation and security would be left apart in these scenarios.

The current state of the art generally lacks of flexible and reconfigurable security solutions whose costs in terms of memory, energy and network performance are well known. The security architecture we propose in this dissertation tries to fill this gap. Our contribution is in proposing PLASA as the evolution of STaR [84]. PLASA integrates the STaR module for secure communications and realizes a security architecture i) easy to integrate, also with legacy applications that have been designed without taking security into account, ii) reconfigurable, both in terms of change of security algorithms in use and in terms of extensibility of PLASA's features by means of new modules, and iii) complete, because it embraces main security requirements (i.e. secure bootstrap, secure communication, key management).

Finally, our work is enriched by a real implementation for real study cases. Data collected from our experiments provides the reader with an idea of the impact of PLASA on performance in terms of memory occupancy, communication overhead and energy consumption. We believe this is an important aspect to be considered when deciding if a security architecture is affordable for resources available on certain devices (e.g. battery-powered sensor nodes). Knowing the cost of security in heterogeneous object systems is vital, because it prevents resource constrained portions of the network from a collapse due to lack of resources to support security.

4.2 PLASA architecture

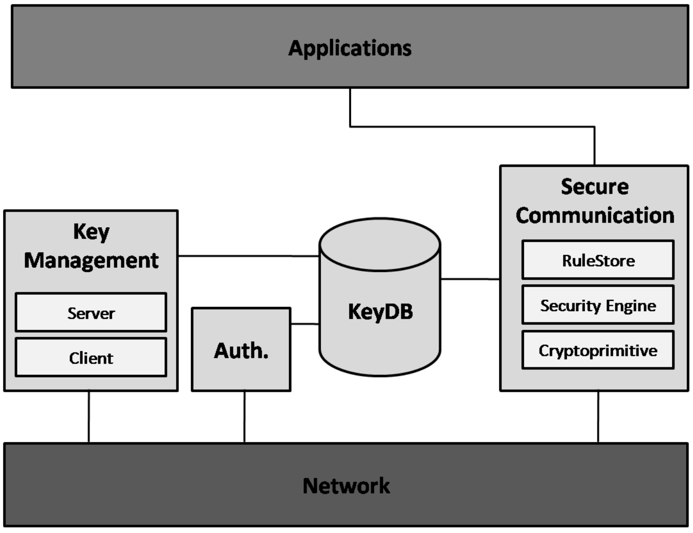

Figure 4.1: PLASA component overview.

PLASA consists of four main components, i) the *Authentication* module, ii) the *KeyDB*, iii) the *Key Management* module, and iv) the *Secure Communication* module. Figure 4.1 shows these modules stay between the application and the communication stack, but the structure can be easily moved between any other layers of the network stack.

The *Authentication* module is responsible for secure bootstrapping of sensor nodes after deployment. The Authentication module objective is to assure that only authorized nodes can access network communication. Since communications are secured by means of cryptographic keys, the Authentication module aims to distribute cryptographic keys to authorized nodes. In order to distribute such keys in a secure manner, the Authentication module relies on a pre-shared secret that is generally loaded into each network member before network deployment.

The KeyDB is organized as a distributed database to store and retrieve cryptographic keys. It acts as a bridge between different modules of PLASA that access the KeyDB to perform different security operations. As shown in Figure 4.1, its design allows to setup, exchange and refresh cryptographic keys without either creating multiple copies of the same key or causing extra overhead due to key exchange from one module to another. The KeyDB module of each node consists of a *Key Table*. Entries of the Key Table have the same structure: i) the *KeyID field*, used to identify the key within the Table, ii) the *key field*, storing the cryptographic key, and iii) the *Flags field*, used to specify some characteristics of the key usage (e.g. pairwise key, group key, key encryption key etc.). Please note that, in our implementation, critical races are automatically avoided by TinyOS. As a consequence, we do not have problems in controlling access to the KeyDB from different modules.

The *Key Management* module is responsible for refreshing or renewing cryptographic keys periodically or on demand when certain events happen. As shown in Figure 4.1, this module includes two submodules: i) the server, acting as a Key Manager that broadcasts cryptographic keys to be renewed according to the Key Management protocol, and ii) the client, receiving updates from the Key Manager and taking appropriate actions. The major part of network nodes that implement the Key management module act as clients, while those implementing the server part are special entities, and in general have no other roles.

4.2.1 STaR module

The *Secure Communication* module is called *STaR* (*Security Transparency and Reconfigurability*). STaR is the heart of PLASA, and extends a preliminary work presented in [84]. STaR secures communications with high adaptivity, thanks to three layers of data abstraction: i) *traffic flows*, ii) *security policies*, and iii) *labels*.

A *traffic flow* is a set of application messages handling the same data or providing the same service. A *security policy* determines what kind of security algorithm is applied by STaR to a certain traffic flow. A *label* determines the mapping between a traffic flow and a security policy. All packets belonging to a given traffic flow can be associated to a common label. Thanks to the label, incoming packets can be unsecured upon being received, according to the security policy associated to the traffic flow they belong to. The STaR component intercepts both incoming and outgoing traffic, segments it into traffic flows, and secures or unsecures them according to the corresponding security policies.

STaR provides transparency of security because exports the same interface of the underlying communication stack. Also, STaR assures reconfigurability by allowing users to dynamically change, enable and disable security policies. STaR modular design allows to i) extend it with any other module, ii) move STaR between any layers of the network stack, iii) adapt its interfaces to any communication paradigm.

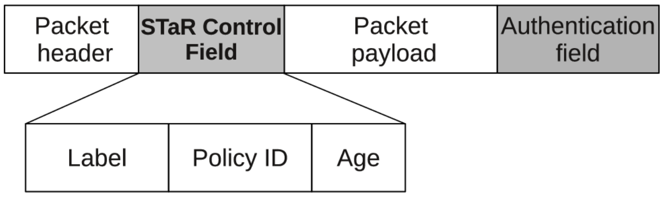

Figure 4.2: Example of packet processed by STaR.

Figure 4.2 shows a packet processed by STaR. Packets arrive at STaR with a header and a payload. Then, STaR builds and inserts the *STaR Control Field* between the header and the payload of the packet. The *STaR Control Field* carries i) the *Label subfield*, which contains the label associated to the traffic flow of the packet, ii) the *PolicyID subfield*, which contains the identifier of the security policy associated to the label, and iii) the *Age subfield*, which specifies the age of the mapping of the label to the policy ID. According to the security policy in use, a packet can have its payload encrypted, can be appended with an authentication trailer, or can have both payload and authentication trailer encrypted.

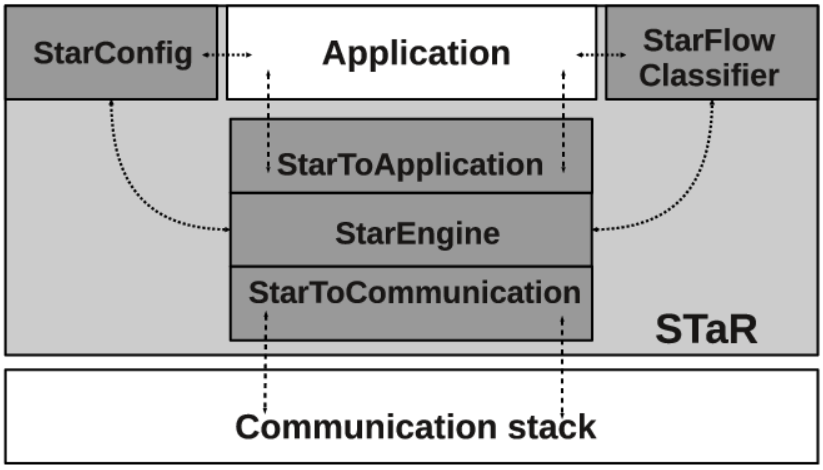

Figure 4.3: STaR architecture.

As shown in Figure 4.3, the STaR component consists in 5 sub-components, namely *StarConfig*, *StarFlowClassifier*, *StarToApplication*, *StarToCommunication*, and *StarEngine*. The *StarFlowClassifier* classifies packets into traffic flows, and determines the associated label. Its implementation consists in a mapping of traffic flows to labels. Application traffic flows can be divided in many ways (e.g. active message types, destination address etc.). The *StarConfig* component allows users to dynamically enable/disable security policies, and change their association to traffic flows, thus providing reconfigurability at runtime. The *StarToApplication* component provides the application with the same communication interface exported by the communication stack. The *StarToCommunication* component makes it possible to connect STaR to the underlying communication stack. The *StarEngine* component encapsulates the mechanism to process security policies and the needed security algorithms.

STaR modularity eases the porting of STaR onto different communication stacks. A different communication stack requires to customize the *StarToCommunication* and *StarToApplication* components, while other components remain unmodified. Although the application developer is not required to change the application code and/or behavior, she has certain obligations in order to exploit STaR, namely i) implementation of security policies; ii) traffic segmentation; iii) association of security policies to traffic segments; and, iv) STaR initialization.

STaR security services

As described in Section 4.2.1, STaR relies on packet labels to protect multiple traffic flows at the same time. All packets belonging to a given packet flow can be associated to a common label, and secured before being transmitted, according to a specified security policy. Incoming packets can be unsecured upon being received, according to the security policy associated to the traffic flow they belong to.

STaR is responsible for securing/unsecuring packets and mapping flow labels into security policies. These tasks are totally transparent to the application. The application can still rely on the original communication interface provided by the available communication stack, and does not require to be modified. In order to manage associations between traffic flows and security policies, STaR maintains two tables: i) a *Security Policy Table* (*SPT*), and ii) a *PolicyDB*.

The *SPT* is formatted as follows. The *Label* field is one byte in size. Thus, STaR manages up to 256 different traffic flows at the same time. The *PolicyID* field specifies the security policy to be adopted for a given traffic flow. *PolicyID* entries in the *SPT* refer to security policies specified in the *PolicyDB* by the specific STaR implementation. The *Active* field indicates whether the security policy associated to a given label has to be applied or not to packets belonging to such traffic flow. The *Active* field is set to TRUE by default in all entries. *SPT*s of all network nodes are supposed to be initialized in the same way at the network startup.

The *PolicyDB* is formatted as follows. *PolicyID* values in the *PolicyDB* have to match *PolicyID* entries of the *SPT* to correctly retrieve the security policy implementation provided by STaR. The *EntryPoint* field contains a reference to the code section which implements the policy (e.g. a C++ function pointer).

STaR communication support

The following communication functions are provided.

```
bool send(packet, size);
```

Provide the packet *packet* of size *size* to STaR. Return TRUE in case of success, FALSE otherwise.

```
bool receive(packet, size);
```

Provide the application with the packet *packet* of size *size* coming from STaR. Return TRUE in case of success, FALSE otherwise.

```
int retrieveLabel(packet);
```

Return the label associated to the traffic flow which the packet *packet* belongs to.

```
Policy retrievePolicy(label);
```

Return the security policy associated to *label*. Return an error code if the *Active* field in the *SPT* is set to FALSE, or the policy is not present in the *PolicyDB*.

The application developer must determine the best security policy to protect each traffic flow, and bind each one of them to a specific label value. Specifically, the *retrieveLabel* function must implement the criteria according to which it is possible to infer which traffic flow a given packet belongs to.

Packet P is transmitted according to the following steps. The application provides STaR with packet P, through the *send* function. Then, STaR retrieves the label L associated to packet P through the *retrieveLabel* function, and the associated security policy SP through the *retrievePolicy* function. Then, STaR builds a one byte field, fills it with the label L, and inserts it between the header and the payload of packet P. Then, packet P is secured, according to the security policy SP. Finally, STaR provides the secured packet P to the communication stack, to deliver it to the recipient node(s). The label must never be encrypted, in order to assure that packet P is correctly unsecured at the recipient side. However, the label can be authenticated, in order to guarantee that it has been actually generated by the STaR component. Figure 4.4 shows an outgoing packet processed by STaR.

Packet P is received according to the following steps. STaR receives the secured packet P from the communication stack, and retrieves the label L from the additional label, which can then be removed. Then, STaR retrieves the security policy SP associated to label L, and unsecures packet P, according to SP. Finally, STaR provides the unsecured packet to the application. Figure 4.5 shows an incoming packet processed by STaR.

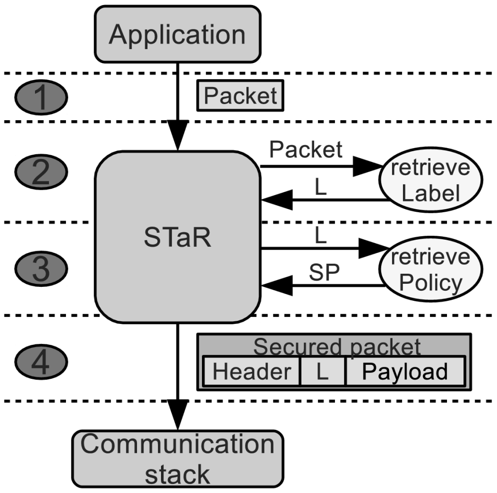

Figure 4.4: Outgoing packet processing.

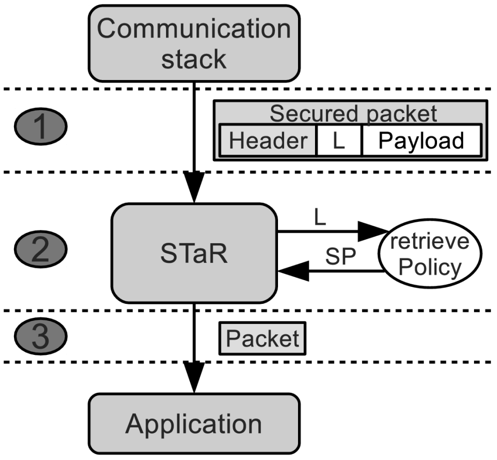

Figure 4.5: Incoming packet processing.

STaR dynamic reconfiguration

STaR dynamic reconfiguration exploits the injection in the network of a small *STaR control packet* that changes the security policy associated to a certain label.

Figure 4.6 shows the format of a *STaR control packet*. The *Reconfiguration info* payload consists in an *Action* field that identifies the kind of action required by the control packet. Following fields represent the *STaR Control Field* of packets belonging to the flow whose policy has to be enabled, disabled or changed, according to the Action field. Note that this packet carries its own *STaR Control Field*, because it has to be at least authenticated. In case the *STaR control packet* is not authenticated, an

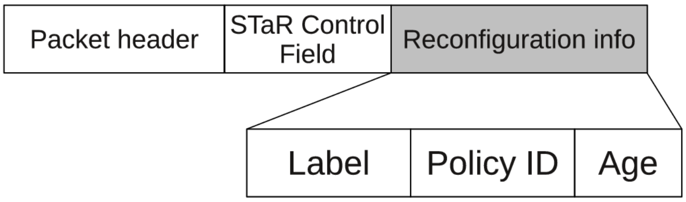

Figure 4.6: STaR control packet format.

adversary might inject fake packets to force STaR to associate the *no_security* policy to all labels, forcing all network nodes to switch to unsecured communication.

Even if the *STaR control packet* is a very short packet which is sent broadcast, it may happen that a network node misses it, so missing the policy reconfiguration. In order to avoid this, STaR provides a *policy reconfiguration recovery procedure*. Every time a node receives a packet, STaR checks whether the label/policy ID association carried within the *STaR Control Field* matches the association specified in the *SPT* for the same label. In case of mismatch, it compares the *Age* subfield carried within the *STaR Control Field*, with the one in the *SPT*. If the *SPT Age* field is greater than the *Age* value of the packet, it means that the sender of the packet missed the *STaR control packet*. In this case the receiver retransmits a *STaR control packet*, to allow the sender to update the *SPT*. Otherwise, if the *Age* value of the packet is greater than the *SPT Age* field, it means that the receiver of the packet missed the *STaR control packet*. In this case, the receiver just updates its *SPT* entry according to information carried within the *STaR Control Field*.

STaR control packets retransmission is broadcast by default, because the STaR-Communication module relies on communication paradigms that are logically broadcast. If the STaRCommunication module supports unicast communication, STaR modular structure allows to substitute the module that implements policy reconfiguration recovery procedure by sending a unicast packet, thus avoiding broadcasts. Please note that choosing broadcast communication is the most feasible way to make STaR as general as possible, while STaR modularity allows to customize the component to achieve better performance.

The following three functions are implemented in the StarConfig module and manage policy update procedures.

```
int enablePolicy(label);
```

Search the *SPT* entry related to label *label*. If a match is found, set to TRUE the corresponding *Active* field. Once the policy is active, STaR starts applying such a security policy to all packets belonging to the traffic flow associated to label *label*. Return zero in case of success, an error code otherwise.

```
int disablePolicy(label);
```

Search the *SPT* entry related to label *label*. If a match is found, set to FALSE the corresponding *Active* field. Then STaR stops applying such a security policy to all packets belonging to the traffic flow associated to label *label*. Return zero in case of success, an error code otherwise.

```
int changePolicy(label, newPolicy);
```

Write *newPolicy* in the *PolicyID* field of the *SPT* entry related to label *label*. The *Active* field of the *SPT* entry remains unchanged. Return zero in case of success, an error code otherwise.

Thanks to the STaR configuration interface, the application is allowed to change security settings at runtime. Also, STaR can dynamically change its behavior even in software platforms which does not allow for dynamically loading/unloading modules, such as TinyOS [3]. This is possible by filling all the implemented *SPT* entries and activating/deactivating them, by simply calling the configuration interface functions.

If the operating system allows for changing some program modules at runtine, it is possible to add and remove policies and traffic flows, in order to match new security requirements more effectively.

The following four functions can be implemented only if the operating system allows for dynamically changing the program loaded on sensor nodes.

```
void addPolicy(PolicyID, EntryPoint);
```

Add the policy identified by *PolicyID* to the *PolicyDB*. *EntryPoint* specifies the code section to be executed to apply the specified policy.

```
void removePolicy(PolicyID);
```

Remove the policy identified by *PolicyID* from the *PolicyDB*.

```
void addFlow(label);
```

Add a flow with ID *label* to the *SPT*. Firstly, verify it is not a copy of another flow, then set the *PolicyID* field to UNDEFINED and the *Active* field to FALSE. These fields will be set by a policy association.

```
void removeFlow(label);
```

Remove the flow identified by *label* from the *SPT*.

4.2.2 Authentication module

The Authentication module maintains the *Neighbors Table* to keep track of neighbors discovered by each network node during the bootstrapping phase. The structure of this table is the following. The *nodeID* field is the unique ID of the discovered neighbor, the *Trusted* flag indicates whether the cryptographic keys associated to him have been successfully authenticated or not, finally, the *keyID* is used to access he *Key Table* to retrieve keys from the KeyDB module.

The only interface function of the Authentication module is the *nodeInit* function. This function initializes the node with its unique identifier. Generally, the cryptographic material required by the secure bootstrapping protocol is generated starting from a pre-shared secret installed on the KeyDB module.

```
int nodeInit(nodeID);
```

Initialize the node with the identifier provided as input by *nodeID* and generate cryptographic material needed by the protocol. Return zero in case of success, an error code otherwise.

4.2.3 KeyDB module

The KeyDB module maintains a set of keys within the *Key Table*. To assure the maximum flexibility, we implemented the functions to get and set cryptographic keys providing the *Flags* input variable. The right usage of cryptographic keys is managed comparing the Flags parameter provided as input and the Flags field specified in the *Key Table*.

```
void setKey(KeyID, Key, Flags);
```

Associate the pairwise key *Key* to the entry identified by *KeyID*, specifying the provided *Flags* in the corresponding field of the Key Table.

```
Key getKey(KeyID, Flags);
```

Access the Key Table looking for the provided *KeyID*. If *Flags* provided as input match the corresponding Flags field, return the key *Key*, otherwise return an error code.

4.2.4 Key management module functions

This module has not specific data structures because it relies on the KeyDB module to store, retrieve, refresh or delete cryptographic keys. The operations of the client submodule are different from those of the server submodule.

Server operations are described as follows.

```
int initKey(nodeID);
```

Trigger the initialization of node *nodeID* with some cryptographic material generated according to the protocol. Return zero in case of success, an error code otherwise.

```
int updateKey(nodeID);
```

Trigger a key refresh of node *nodeID* with some cryptographic material generated according to the protocol. Return zero in case of success, an error code otherwise.

Client operations are described as follows.

```
int requestKey(keyID);
```

Ask to the Key Manager to (re)send the key identified by *keyID*, to recover from key losses. Return zero in case of success, an error code otherwise.

```
int updateKey(keyID);
```

Manages a key refresh request from the key manager by updating the key identified by *keyID*, according to the protocol. Return zero in case of success, an error code otherwise.

4.3 PLASA performance evaluation

We implemented the PLASA components [72] for TinyOS 2.x, which is currently available at [3], with reference to the Tmote Sky motes [4] and the CC2420 chipset [31]. As to security algorithms supported by STaR, we have implemented i) the *Skipjack* encryption module [85], ii) an authentication algorithm that uses SHA-1 [86] to compute the message digest and then symmetric key encryption for authentication (hereafter SHA-1-based authentication, for short), and iii) a module providing hardware security by means of the CC2420 chipset. The CC2420 chipset provides security primitives for both encryption and authentication relying on the AES encryption algorithm [29].

4.3.1 Memory occupancy

The amount of ROM memory available on Tmote Sky motes is 48 KB, and may represent a severe constraint while dealing with complex modules like those composing PLASA. In order to evaluate memory consumption on Tmote Sky motes, we considered the impact of single modules on TinyOS image size.

	Memory occupancy (bytes)	**Memory occupancy (%)**
Authentication module	2720	5.67
STaR	1610	3.35
StarEngine (Skipjack)	1748	3.64
StarEngine (SHA-1-based authentication)	3442	7.17
StarEngine (HW AES-128)	1444	3.01
KeyDB module	198	0.41
Rekeying client	288	0.60

Table 4.1: Detailed memory occupancy.

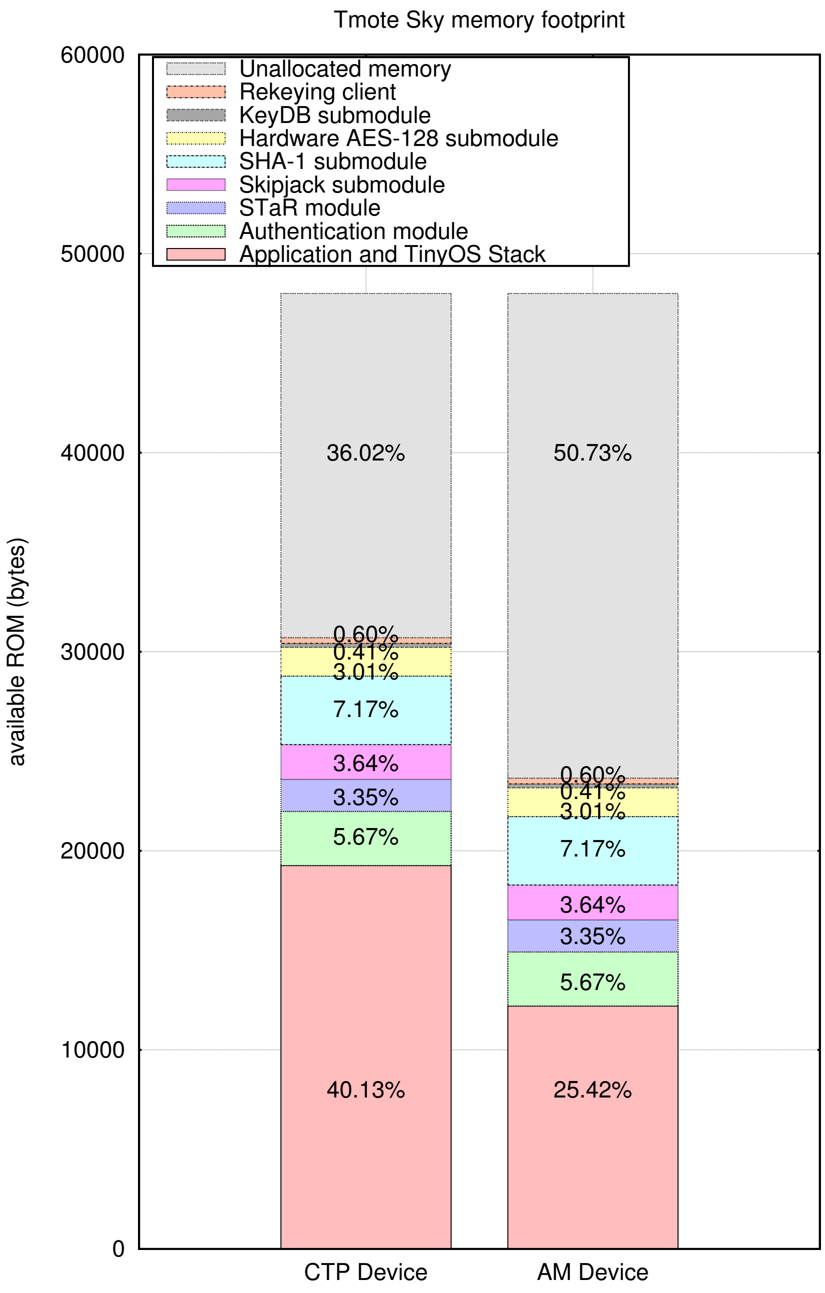

Figure 4.7: PLASA memory footprint.

Table 4.1 and Figure 4.7 show the percentages of Tmote Sky ROM occupied by the RadioCountToLeds application, comparing the CTP routing protocol implementation and the ActiveMessage implementation. If we sum the contributions of the *STaR*, *Skipjack*, *SHA-1* and *Hardware AES-128* modules, we observe that STaR implementation totally requires the 17.17% of the overall memory available on a Tmote Sky mote. Similarly, if we sum the contributions of all PLASA modules, we observe that a full implementation of PLASA requires the 23.85% of the overall memory available on a Tmote Sky mote. The application together with the TinyOS stack using CTP routing requires the 40.13% of the available memory, and leaves the 36.02% of 48 KB still available for other uses. Similarly, the application together with the TinyOS stack using ActiveMessages requires the 25.42% of the available memory, and leaves the 50.73% of 48 KB still available for other uses. Please note that reported percentages are referred to the amount of Tmote Sky memory required by single PLASA modules and is not influenced by the communication paradigm (ActiveMessages or CTP). We believe that this amount of memory is reasonable with respect to the available memory. Also, our study case shows that PLASA modular implementation allows for saving memory by loading only a few of the available modules, provided that it is well known what modules are needed.

4.3.2 PLASA communication and processing overhead

In our analysis, we assumed that PLASA is operating on top of the 2.4 GHz physical layer, with a 250 Kb/s bit rate [31]. We modeled the impact of security considering two main aspects: i) the network performance degradation due to security processing and transmission of extra bytes in each packet to guarantee security, and ii) the extra energy consumption, due to extra processing and extra transmissions. We evaluated the security processing overhead by means of experiments, while the extra transmission overhead and extra energy consumption have been computed analytically.

The *PLASA processing overhead* delay (d_{proc}) is the extra time required by the PLASA module to perform the required security operations. This delay has been evaluated experimentally for each of the involved security features of our PLASA TinyOS implementation. In order to increase the accuracy of our results, we performed 10 repetitions of 20 transmissions for each experiment. Results are averaged over all the different repetitions. We also report the standard deviation we derived from the independent replication method. The transmission overhead (d_{tx}) has been evaluated analytically, considering a 250 Kb/s bit rate [31]. Specifically, we have considered the time required to transmit the additional bytes added by PLASA for each of the involved security features and each of its modules.

Table 4.2 provides an overview of $\mathrm{d_{proc}}$ contributions for PLASA modules. For each row in the table, we report the amount of time required by each PLASA module to perform its operations. The value of the secure bootstrapping entry is related to the whole protocol and, since it runs just once after deployment, we consider such a

PLASA module	$\mathbf{d_{proc}}$ (μs)	Standard deviation (μs)
Secure bootstrapping	98804.8	12987.93
STaR	96	0
STaR Skipjack encryption	1217.6	7.16
STaR SHA-1-based authentication	33212.8	30.97
STaR Skipjack encryption and SHA-1-based authentication	34318.4	54.42
STaR hardware AES-128-based encryption and authentication	216	14.22
Rekeying	924.8	22.98

Table 4.2: PLASA $\mathrm{d_{proc}}$ contributions for PLASA modules.

contribution affordable. The $\mathrm{d_{proc}}$ contributions of STaR software implemented algorithms (from third to fifth rows) are due to the algorithms and are not due to STaR. On the other hand, the contribution of STaR reported in second line and is just the delay added by the labelling mechanism. The $\mathrm{d_{proc}}$ contribution of STaR hardware implemented algorithm (sixth row) is considerably lower than those of software implementations, and can substitute software algorithms when the processing delay cannot exceed some timing constraints. Finally, we believe that also the value reported for rekeying (seventh row) is affordable.

In order to evaluate the transmission overhead analytically, we have considered the time required to transmit different kinds of packets at different stages of the PLASA secured WSN application, with a bit rate of 250 Kb/s.

PLASA module	$\mathbf{d_{tx}}$ (μs)	Extra bytes
Secure bootstrapping	2976	93
STaR	32	1
STaR Skipjack encryption	256	8
STaR SHA-1-based authentication	640	20
STaR Skipjack encryption and SHA-1-based authentication	896	28
STaR hardware AES-128 encryption and authentication	512	16
Rekeying	1504	47

Table 4.3: PLASA $\mathrm{d_{tx}}$ contributions for PLASA modules.

Table 4.3 provides an overview of the transmission overhead, in different PLASA execution phases. As already observed for the processing overhead, the considerable delays of the STaR encryption and authentication policy are due to the standard SHA-1-based authentication output size, which is 20 bytes long. In fact, the actual PLASA contribution to the transmission delay is just 32 $\mu\mathrm{s}$, that is the time required to transmit the one byte *STaR Control Field* added to the original packet. We believe that this delay is affordable, since it is due to the increase of just one byte to the original packet size. Considering a 250 Kb/s bit rate [31], the transmission overhead increase for the SHA-1-based authentication field can be evaluated analytically as $\mathrm{d_{tx}} = \frac{20 \cdot 8}{0.250} = 640\mu\mathrm{s}$. If the application developer finds unaffordable such an increase in the transmission delay, it is possible to define security policies which truncate the hashing field to 4, 8 or 16 bytes in size. Considering a 250 Kb/s bit rate, this would save 512 $\mu\mathrm{s}$, 384 $\mu\mathrm{s}$ and 256 $\mu\mathrm{s}$ during packet transmission, respectively. Hash field truncation is a widely adopted method in WSNs, because it allows for increasing performance without serious risks of collisions [7, 41]. Also in this case, the value of the secure bootstrapping entry is related to the whole protocol and, since it runs just once after deployment, we consider such a contribution as affordable. The $\mathrm{d_{tx}}$ contribution of rekeying packets is not negligible. However, these packets are transmitted for periodic key refresh only. The rekeying period can be tuned in order to trade-off security and performance.

4.3.3 PLASA energy consumption

Energy consumption has been evaluated analytically, considering single energy contributions. We expressed energy consumption contributions as $\mathcal{E}_\mathrm{i} = \mathcal{P}_\mathrm{i} \cdot \mathrm{d_i}$. Let $\mathrm{d_i}$ be the delay due to the considered operation i. Let $\mathcal{P}_\mathrm{i} = \mathrm{V_i} \cdot \mathrm{I_i}$ be the single power contribution, i.e. the product between voltage and current of the Tmote sky processing unit and CC2420 components. The values of absorbed current referring to the Tmote sky processing unit and CC2420 components are taken from the respective datasheets [4, 31]. The only exception is the value of the absorbed current during hardware security operations that has been taken from [17].

Table 4.4 provides an overview of energy contributions for PLASA modules. Considerable contributions in per packet energy consumption for both processing and transmission are due to the standard encryption and authentication algorithms. Note that, while hardware security improves network performance, it has comparable values from the energy consumption standpoint. This is due to the fact that the CC2420 chipset consumes more current than the Tmote Sky processing unit. On the other hand, since hardware security is faster than software security, the reduced time of usage balances the increase of current consumption, allowing us to obtain energy consumptions that are comparable in hardware and software security operations.

The total STaR contribution to per packet energy consumption (second row) is just $103.68 + 1002.24 = 1105.92$ nJ.

Similarly to network performance case, the value of the secure bootstrapping entry is related to the whole protocol and, since it runs just once after deployment, we

PLASA module	**Processing** $\mathcal{P}_{\text{proc}} = 1.08\text{mW}$		**Transmission** $\mathcal{P}_{\text{tx}} = 31.32\text{mW}$	
	$\text{d}_{\text{proc}}(\mu\text{s})$	$\mathcal{E}_{\text{proc}}$ (nJ)	$\text{d}_{\text{tx}}(\mu\text{s})$	$\mathcal{E}_{\text{tx}}$ (nJ)
Secure bootstrapping	98804.8	106709.18	2976	93208.32
STaR	96	103.68	32	1002.24
STaR SkipJack encryption	1217.6	1315.01	256	8017.92
STaR SHA-1-based authentication	33212.8	35869.82	640	20044.8
STaR SkipJack encryption and SHA-1-based authentication	34318.4	37063.87	896	28062.72
Rekeying	924.8	2.81	1504	47105.28

PLASA module	**Processing** $\mathcal{P}_{\text{proc}} = 38.14\text{mW}$		**Transmission** $\mathcal{P}_{\text{tx}} = 31.32\text{mW}$	
	$\text{d}_{\text{proc}}(\mu\text{s})$	$\mathcal{E}_{\text{proc}}$ (nJ)	$\text{d}_{\text{tx}}(\mu\text{s})$	$\mathcal{E}_{\text{tx}}$ (nJ)
STaR hardware AES-128 encryption and authentication	216	8238.24	512	16035.84

Table 4.4: PLASA energy consumption contributions.

consider a total contribution of $106709.18 + 93208.32 = 199917.50$ nJ as affordable. The energy contribution of rekeying packets is $2.81 + 47105.28 = 47108.09$ nJ. We believe that this value is affordable because this packet is transmitted for periodic key refresh only. Also, the rekeying period can be tuned in order to trade-off security and energy consumption.

4.4 PLASA integration study case

The modular structure of PLASA allows for integrating the whole architecture or single modules in any network stack and application scenario. A user who wants to use PLASA needs to integrate PLASA's modules into the program running on each component of the network. Since PLASA is highly modular, it is possible to evaluate case by case what modules have to be instantiated, thus saving some memory on resource constrained network components (e.g. sensor nodes). Although the application developer is not required to change the application code and/or behavior, she has certain obligations in order to exploit PLASA, namely i) instantiation and linking of needed PLASA modules, ii) PLASA modules initialization to rely on the right communication paradigm, iii) if STaR is present, traffic segmentation into data flows and association of security policies to traffic segments.

In the following, we report PLASA interface functions and data structures, providing some information about the modules they belong to. We put emphasis on the adaptability and modularity of PLASA presenting two study cases. The first use case demonstrates that the whole PLASA architecture can be integrated into a COS application. Here we use all PLASA's modules to secure a two-phase sensed data collection application. Also, we will use STaR to protect communications involving different layers of the network stack.

4.4.1 PLASA to protect a COS from scratch

This study case integrates all PLASA modules with an application with many sender nodes that just send or forward messages to a destination base station. We considered a real data collection application scenario that relies on cooperation between an *Unmanned Aerial System* (*UAS*) and a WSN. The application scenario and the mobility model of the UAS are very similar to those reported in [87]. The application involves three types of actors: i) *Sensor node* (*SN*) which senses the environment and sends data to an aggregator, ii) *Cluster Head* (*CH*) which collects data from SNs and aggregates them, and iii) *Unmanned Aerial System* (*UAS*) whose flight plan is to fly over CHs. The UAS has an on-board sensor that communicates with CHs to collect aggregated data.

Since the entire traffic in the ground network is forwarded to the CH, the application design fits well the *Collection Tree Protocol* (*CTP*) routing protocol [23]. CTP creates a routing tree whose root is the CH. STaR can be transparently integrated with both CTP routing tree building phase and CTP communication paradigm. Also, we include the *PLASA Authentication*, *KeyDB* and *Key Management* modules to have a complete PLASA implementation.

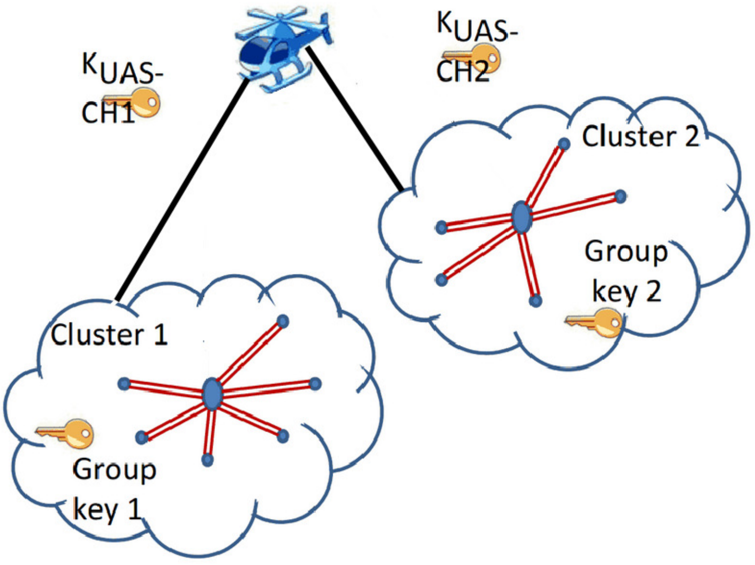

Figure 4.8: PLASA secured COS scenario.

Figure 4.8 shows the application scenario we are considering. The application consists of six phases: i) the *secure bootstrapping phase*, ii) the *discovery phase*, iii) the *routing tree building phase*, iv) the *data aggregation phase*, v) the *data collection phase*, and vi) the *rekeying phase* . Phases i), ii) and iii) take place once, after network deployment. However, phase ii) may be repeated after changes in the network topology. Phases iv) and v) repeat in loop during the entire network lifetime. Finally, phase vi) periodically overlaps phases iv) and v) to refresh cryptographic keys.

The *secure bootstrapping phase* just performs the secure bootstrapping algorithm integrated in the *PLASA Authentication module*, supported by the *PLASA KeyDB module*. At the end of this phase, all network members share a group key that is stored in the *PLASA KeyDB module*, and is used to secure messages exchanged during the *discovery phase* and the *routing tree building phase*.

The *discovery phase* is in its turn divided in two subphases. Firstly, each CH advertises himself and SNs collect information to decide the best CH to be associated with. Then, each SN chooses its CH and advertises this choice broadcasting a message. At the end of this phase, clusters are created. Two types of messages are exchanged in this phase: i) *Advertisements* sent by the CHs to SNs, to advertise themselves as CHs, and ii) *Associations* sent by SNs to the CH they are associated with. Communications are secured by means of *STaR*, using the key exchanged in the *secure bootstrapping phase*.

Advertisements and *Associations* are logically mapped onto two different data flows. Thus, they can be associated to two different security policies to secure them appropriately. The threat for both Advertisements and Associations is represented by an adversary claiming to be a regular CH or a regular SN, respectively. A malicious CH may create a *Denial of Service*, because SNs associate to the malicious CH, while the regular CH fails in creating a cluster. On the other hand, a malicious SN may associate to a CH becoming member of a regular cluster, possibly injecting fake messages in the cluster. We authenticate both messages by means of the SHA-1-based authentication algorithm. Since CH and SNs share the cryptographic key used to create the *Message Authentication Code* (*MAC*), once they correctly verify the MAC of messages, they can assume messages have been sent by a regular CH or SN.

The *routing tree building phase* consists in the classical CTP messages exchanges to build the routing tree. Possible threats in this phase is an adversary cheating its *Received Signal Strength Indicator* (*RSSI*) to create a wormhole or advertising itself as the root of the tree, preventing the root node from receiving packets. In order to avoid this, we need routing messages to be authenticated, so that only messages sent by trusted members of the network can contribute to create the routing tree.

Once that the authenticated routing tree has been built, sensor nodes start sending messages to the base station. This messages not only suffer from the same threats we have described for the *routing tree building phase*, but may also carry sensitive data, that should not be overheard by malicious nodes. For this reason, we aim at guaranteeing both authenticity and confidentiality for this kind of messages. Authenticity is provided by the SHA-1-based authentication algorithm, while confidentiality is assured by the Skipjack encryption algorithm [85]. Thanks to encryption, even if malicious nodes intercept data messages, they cannot retrieve information carried within them.

During the *data aggregation phase*, SNs sense data and stores them. Periodically, each CH asks SNs belonging to his cluster for sensed data and collects their answers. Two types of messages are exchanged in this phase: i) *Beacons* sent by the CH to SNs, to ask for data, and ii) *Replies* sent by SNs to CH to report sensed data. Finally, each CH aggregates the Replies he receives and creates an *Aggregated data* message. In this phase communications are secured by means of STaR.

Beacons are broadcast messages transmitted by the CH, they do not carry any sensitive information, just inform sensor nodes of the presence of a CH asking for sensed data. The threat for such a message is represented by an adversary claiming to be a regular CH. This may create a *Denial of Service*, because sensor nodes send data to the malicious CH, while the regular CH does not get any data. This Denial of Service attack can be avoided guaranteeing that *Beacons* are transmitted by the regular CH. We authenticate *Beacons* with the SHA-1-based authentication algorithm.

Replies are unicast messages carrying information about data sensed by a sensor node of the network. These messages not only suffer from the same threat we described for *Beacons*, but may also carry sensitive data. To prevent these messages from being overheard by the aforementioned malicious CH, we guarantee both authenticity and confidentiality with the SHA-1-based authentication algorithm and the Skipjack encryption algorithm, respectively.

During the *data collection phase*, every time the UAS flies over a CH, sends a message asking for Aggregated data. Once the CH receives the request, answers with the Aggregated data message he has prepared. Two types of messages are exchanged in this phase: i) *Beacons* sent by the UAS on-board sensor to CHs, to ask for data, and ii) *Aggregated data* sent by the CH to the sensor node which stays on the UAS, when the UAS is close enough to allow communication between the CH and the on-board sensor node. In this phase, communications are secured by means of STaR. *Beacons* are logically the same message exchanged during the *secure data aggregation phase*, they only differ in sender and receiver. As a consequence, these messages are authenticated with the same method described above. *Aggregated data* suffer from the same threats as *Replies*, so we applied the same policy to guarantee both authenticity and confidentiality. However, these messages have also some time constraints, so we use the CC2420 microcontroller [31] to rely on a STaR security policy for high-performance hardware encryption and authentication.

Finally, we have the *rekeying phase*, performed by the *PLASA Rekeying module*, supported by the *PLASA KeyDB module*. In long lived networks, an adversary might record messages exchanged within the network. Thus, she would aim at recovering cryptographic keys by cryptanalyzing secured data or building a dictionary for a dictionary attack. In order to avoid this, the Key Manager wakes up the *PLASA Rekeying module* by periodically broadcasting messages to force network nodes to update their cryptographic keys. If the rekeying period is well tuned, recording message for cryptanalysis or dictionary attacks, turns out to be useless.

PLASA integration

To use PLASA to protect a COS, we performed some application-specific operations. We instantiated *PLASA KeyDB* and *PLASA Authentication* modules and wired those components to the TinyOS Boot module, so that these components start before any other application module. Also, we instantiated three *STaR modules*. Since STaR is a *generic* nesC module [88], it can have many instances without any interference between them. Note that STaR tables of one instance are completely independent from those of the other instance. Thus, they can be filled with different policies.

The first STaR instance has to protect routing messages. This module has its *StarToCommunication* and *StarToApplication* components wired to modules responsible for building the CTP tree. We relied on hardware authentication primitives provided by the CC2420 chipset [31] to achieve high efficiency when securing routing messages. On the other hand, we used software encryption and authentication policies to save some energy when securing exchange of data messages.

The second and third STaR instances have to protect application data. The second instance has its *StarToCommunication* and *StarToApplication* components wired to communication modules responsible for Beacons and Replies exchange. The third one has its *StarToCommunication* and *StarToApplication* components wired to modules responsible for UAS requests and Aggregated data exchange.

We segmented traffic into data flows developing the *StarFlowClassifier* module. We referred to the TinyOS Active Message types [3] to recognize the message types described before. Then, in order to associate security policies to traffic segments, we parsed each application packet to retrieve the Active Message type field. Finally, we used the *StarFlowClassifier* module to instruct STaR to use the *authentication_only* Label for Beacons and the *encryption_and_authentication* Label for Replies and Aggregated Data packets.

To complete the integration process, we wired TinyOS modules so that STaR stays between the data application modules and the rest of communication stack. Developing the *StarFlowClassifier* module we accomplish traffic segmentation into data flows. We used the *StarFlowClassifier* module to instruct STaR to use the *authentication_only* label for Beacons and the *encryption_and_authentication* label for Replies and Aggregated Data packets.

5

Conclusion

In this Ph.D. dissertation, we have considered and evaluated the importance of security in Wireless Sensor Networks (WSNs). We have focused on the impact of security on performance, that is a fundamental aspect when we consider a network involving resource-constrained devices such as sensors. Security operations are generally resource greedy in terms of memory and computation capabilities. On the contrary, sensors are limited in terms of memory capacity, computational capabilities, and also battery lifetime.

As a consequence, any security solution for WSNs needs to keep into account the cost in terms of memory, computation and battery lifetime, so making it possible to trade-off effectiveness and efficiency. Knowing in advance the costs of different security policies allows to switch from a security policy to another for a better response to environmental changes or attacks. Thus, it is easier to provide applications with the security features they need, also increasing users' trust in secure applications.

In this Ph.D. dissertation, we have addressed the above mentioned issues, and provided the following contributions to WSN security.

- Analysis and performance evaluation of IEEE 802.15.4 security services. We have shown that security mechanisms and options as provided by the standard cause the increase of frame length (communication overhead) and require additional computations (computing overhead) for security processing. We have shown the relationship between the computing and communication overhead and the security parameters, namely security level and key identification mode. In addition, we have shown that the greatest cost has to be paid when we switch from unsecured traffic to secured traffic. However, when using hardware-based cryptography, the chosen security service has little, or even negligible, impact on performance. On the contrary, when using software-based cryptography, the chosen security service and the payload size has a considerable impact on performance.
- An IEEE 802.15.4 compliant countermeasure against selective Denial of Service. We have presented and discussed *Selective Jamming Resistant GTS* (*SJRG*), our standard-compliant solution to the GTS-based selective jamming attack, able

to cope with both the intelligent and the sniper attacks. Both attacks aim at disrupting communications by selectively jamming contention free Slots. SJRG relies on IEEE 802.15.4 security services, and provides a two steps countermeasure against selective jamming that reduces the attack success rate, forcing the attacker to pick the target Slot in a random way. We have implemented SJRG for the TinyOS platform on Tmote Sky motes, and evaluated it on a realistic application scenario. In particular, we have shown that i) the attack success rate can be reduced to 13.8%; ii) SJRG results in an additional memory consumption which is definitely affordable; iii) the network performance degradation due to SJRG is practically negligible; and iv) the per packet energy consumption is affordable and mostly due to the IEEE 802.15.4 security sublayer contributions.

- A modular security architecture for Cooperating Objects Systems, namely PLASA. PLASA is designed to be integrated in heterogeneous objects application scenarios. It provides secure communications, key management and secure bootstrapping implemented in a transparent, reconfigurable and modular architecture. The secure communication module, namely STaR, has been designed from scratch and presents some interesting characteristics. It protects multiple traffic flows at the same time, according to different security policies. PLASA is transparent to the application, which can rely on the same communication interface already in use. Also, PLASA is modular because it is possible to integrate an application only with some modules of the whole PLASA architecture. We have considered our implementation of PLASA for Tmote Sky motes, and provided a performance evaluation in terms of memory occupancy, communication overhead, and energy consumption. Our results show that PLASA is efficient as well as affordable even in the considered resource scarce hardware platform. In fact, the heaviest impact on performance is due to the adopted standard security algorithms, and not to the presence of PLASA.

We believe that these contributions represent a valid tool to provide WSNs with security features that guarantee network sustainability, and hope they will be a first step towards further achievements in this field.

References

1. Cardenas A.A., Roosta T., Sastry S., “Rethinking security properties, threat models, and the design space in sensor networks: a case study in SCADA systems,” *Ad Hoc Networks*, vol. 7, no. 8, pp. 1434–1447, 2009.
2. Daidone R., Dini G., Tiloca M., “Open Source Toolset for IEEE 802.15.4 and ZigBee,” 2010. [Online]. Available: http://www.open-zb.net/
3. TinyOS Working Group, “Tinyos home page,” http://www.tinyos.net/, 2012. [Online]. Available: http://www.tinyos.net/
4. Moteiv Corporation, “Tmote iv low power wireless sensor module,” Nov 2006. [Online]. Available: http://www.snm.ethz.ch/snmwiki/pub/uploads/Projects/tmote_sky_datasheet.pdf
5. Sokullu R., Dagdeviren O., Korkmaz I., “On the IEEE 802.15.4 MAC Layer Attacks: GTS Attack,” in *Proceedings of the 2nd International Conference on Sensor Technologies and Applications, SENSORCOMM ’08*, Aug 2008, pp. 673–678.
6. Sokullu R., Korkmaz I., Dagdeviren O., “GTS Attack: An IEEE 802.15.4 MAC Layer Attack in Wireless Sensor Networks,” *International Journal On Advances in Internet Technologies*, vol. 2, no. 1, pp. 104–114, 2009.
7. IEEE, *IEEE Std. 802.15.4-2006, IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, Institute of Electrical and Electronics Engineers, Inc., New York, Sep 2006.
8. Lee J.S., “Performance evaluation of IEEE 802.15.4 for low-rate wireless personal area networks,” *IEEE Transactions on Consumer Electronics*, vol. 52, no. 3, pp. 742–749, Aug 2006.
9. Mišić J., Shafi S., Mišić V.B., “The impact of MAC parameters on the performance of 802.15.4 PAN,” *Ad Hoc Networks*, vol. 3, no. 5, pp. 509 – 528, 2005.
10. Ramachandran I., Das A.K., Roy S., “Analysis of the contention access period of IEEE 802.15.4 MAC,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 3, no. 1, Mar 2007.
11. Chen F., Xiaolong Y., German R., Dressler F., “Performance impact of and protocol interdependencies of IEEE 802.15.4 security mechanisms,” in *Proceedings of the 6th IEEE International Conference on Mobile Adhoc and Sensor Systems MASS ’09*, 2009, pp. 1036–1041.
12. Sastry N., Wagner D., “Security considerations for IEEE 802.15.4 networks,” in *Proceedings of the 3rd ACM workshop on Wireless security WiSe ’04*. New York, NY, USA: ACM, 2004, pp. 32–42.

13. Xiao Y., Chen H.H., Sun B., Wang R., Sethi S., "MAC security and security overhead analysis in the IEEE 802.15.4 wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, pp. 81–81, Apr 2006.
14. Chang C.C., Nagel D.J., Muftic S., "Balancing security and energy consumption in wireless sensor networks," in *Mobile Ad-Hoc and Sensor Networks*, ser. Lecture Notes in Computer Science, H. Zhang, S. Olariu, J. Cao, and D. Johnson, Eds. Springer Berlin Heidelberg, 2007, vol. 4864, pp. 469–480.
15. Guimaraes G., Souto E., Sadok D., Kelner J., "Evaluation of security mechanisms in wireless sensor networks," in *Proceedings of the Systems Communications, 2005*, 2005, pp. 428–433.
16. Law Y.W., Doumen J., Hartel P., "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 2, no. 1, pp. 65–93, Feb 2006.
17. Lee J., Kapitanova K., Son S.H., "The price of security in wireless sensor networks," *Computer Networks*, vol. 54, no. 17, pp. 2967–2978, Dec 2010.
18. Zhu J., Leina G., Xinfang Z., "Implementation and Time Performance Analysis of Security Suite in LR-WPAN 802.15.4," in *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08*, 2008, pp. 1–5.
19. Alcaraz C., Lopez J., Roman R., Chen H.H., "Selecting key management schemes for WSN applications," *Computers & Security*, vol. 31, no. 38, pp. 956–966, Nov 2012.
20. Dini G., Savino I.M., "S2RP: a Secure and Scalable Rekeying Protocol for Wireless Sensor Networks," in *Proceedings of the 3rd IEEE International Conference on Mobile Adhoc and Sensor Systems MASS '06*, 2006, pp. 457–466.
21. Liu A., Ning P., "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," in *Proceedings of the International Conference on Information Processing in Sensor Networks, 2008. IPSN '08*, 2008, pp. 245–256.
22. ZigBee Alliance, *ZigBee Document 053474r17, ZigBee Specification*, ZigBee Alliance, Jan 2008.
23. Gnawali O., Fonseca R., Jamieson K., Moss D., Levis P., "Collection Tree Protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*, Nov 2009.
24. Hauer J.H., Daidone R., Severino R., Büsch J., Tiloca M., Tennina S., "Poster Abstract: An Open-Source IEEE 802.15.4 MAC Implementation for TinyOS 2.1," in *Proceedings of the 8th European Conference on Wireless Sensor Networks EWSN*, Feb 2011.
25. Zheng J., Lee M.J., Anshel M., "Toward Secure Low Rate Wireless Personal Area Networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 10, pp. 1361 –1373, Oct 2006.
26. Ganesan P., Venugopalan R., Peddabachagari P., Dean A., Mueller F., Sichitiu M., "Analyzing and modeling encryption overhead for sensor network nodes," in *Proceedings of the 2nd ACM International conference on Wireless sensor networks and applications WSNA '03*. New York, NY, USA: ACM, 2003, pp. 151–159.
27. Jinwala D., Patel D., Dasgupta K., "Optimizing the block cipher and modes of operations overhead at the link layer security framework in the wireless sensor networks," in *Proceedings of the 4th International Conference on Information Systems Security ICISS '08*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 258–272.
28. IEEE, *IEEE Std. 802.15.4-2003, IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, Institute of Electrical and Electronics Engineers, Inc., New York, Oct 2003.
29. National Institute of Standards and Technology, "Federal information processing standards publication 197, specification for the advanced encryption standard (aes)," Nov 2001.

30. Passing M., Dressler F., “Experimental performance evaluation of cryptographic algorithms on sensor nodes,” in *Proceedings of the 3rd IEEE International Conference on Mobile Adhoc and Sensor Systems MASS ’06*, 2006, pp. 882–887.
31. Texas Instruments, “Texas instruments cc2420 2.4 ghz ieee 802.15.4 / zigbee ready rf transceiver,” http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2012. [Online]. Available: http://focus.ti.com/lit/ds/symlink/cc2420.pdf
32. “TinyOS security algorithms repository.” [Online]. Available: http://sourceforge.net/projects/tinyossecurity/files/
33. Daidone R., Dini G., Tiloca M., “On experimentally evaluating the impact of security on IEEE 802.15.4 networks,” in *Proceedings of the 2011 International Conference on Distributed Computing in Sensor Systems and Workshops DCOSS ’11*, Jun 2011, pp. 1–6.
34. Camtepe S.A., Yener B., “Key management in wireless sensor networks,” in *Cryptology and Information Security Series, Wireless Sensor Network Security*, ser. The Cryptology & Information Security Series (CISS), J. Lopez and J. Zhou, Eds. IOS Press, 2008, pp. 110–141.
35. Xiao Y., Rayi V.K., Sun B., Du X., Hu F., Galloway M., “A survey of key management schemes in wireless sensor networks,” *Computer Commununications*, vol. 30, no. 11-12, pp. 2314–2341, Sep 2007.
36. Amini F., Khan M., Mišić J., Pourreza H., “Performance of IEEE 802.15.4 Clusters with Power Management and Key Exchange,” *Journal of Computer Science and Technology*, vol. 23, pp. 377–388, 2008.
37. Mišić J., “Cost of secure sensing in IEEE 802.15.4 networks,” *IEEE Transactions on Wireless Communications*, vol. 8, no. 5, pp. 2494–2504, 2009.
38. Dini G., Savino I.M., “LARK: A Lightweight Authenticated ReKeying Scheme for Clustered Wireless Sensor Networks,” *ACM Transactions on Embedded Computing Systems*, vol. 10, no. 4, pp. 1–35, Nov 2011.
39. Dini G., Tiloca M., “HISS: A HIghly Scalable Scheme for Group Rekeying,” *The Computer Journal*, 2012.
40. Zhu S., Setia S., Jajodia S., “LEAP+: Efficient security mechanisms for large-scale distributed sensor networks,” *ACM Transaction on Sensor Networks*, vol. 2, no. 4, pp. 500–528, Nov 2006.
41. Karlof C., Sastry D., Wagner D., “Tinysec: a link layer security architecture for wireless sensor networks,” in *Proceedings of the 2nd International conference on Embedded networked sensor systems SenSys ’04*. New York, NY, USA: ACM, 2004, pp. 162–175.
42. Aziz A., Diffie W., “Privacy and authentication for wireless local area networks,” *IEEE Personal Communications*, vol. 1, no. 1, pp. 25–31, 1994.
43. Perrig A., Stankovic J., Wagner D., “Security in wireless sensor networks,” *Communications of the ACM - Wireless sensor networks*, vol. 47, no. 6, pp. 53–57, Jun 2004.
44. Bougard B., Catthoor F., Daly D.C., Chandrakasan A., Dehaene W., “Energy Efficiency of the IEEE 802.15.4 Standard in Dense Wireless Microsensor Networks: Modelling and Improvement Perspectives,” in *Proceedings of the Conference on Design, Automation and Test in Europe DATE 2005*. IEEE Computer Society, Apr 2005, pp. 196–201.
45. “TinyOS security algorithms repository.” [Online]. Available: http://sourceforge.net/projects/tinyossecurity/files/
46. “Network Simulator Ns2.” [Online]. Available: http://www.isi.edu/nsnam/ns/
47. Anastasi G., Conti M., Di Francesco M., “A Comprehensive Analysis of the MAC Unreliability Problem in IEEE 802.15.4 Wireless Sensor Networks,” *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 52–65, 2011.
48. Koubaa A., Alves M., Tovar E., “GTS allocation analysis in IEEE 802.15.4 for real-time wireless sensor networks,” in *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, Apr 2006.

49. Daidone R., Dini G., Tiloca M., “A solution to the gts-based selective jamming attack on ieee 802.15.4 networks,” *Wireless Networks, Springer*, 2013.
50. Wood A.D., Stankovic J.A., “Denial of service in sensor networks,” *Computer*, vol. 35, no. 10, pp. 54–62, Oct 2002.
51. Raymond D.R., Midkiff S.F., “Denial-of-Service in Wireless Sensor Networks: Attacks and Defenses,” *IEEE Pervasive Computing*, vol. 7, no. 1, pp. 74–81, January–March 2008.
52. Noubir G., Lin G., “Low-power DoS attacks in data wireless LANs and countermeasures,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 7, pp. 29–30, Jul 2003.
53. Lazos L., Liu S., Krunz M., “Mitigating control-channel jamming attacks in multi-channel ad hoc networks,” in *Proceedings of the 2nd ACM conference on Wireless network security WiSec ’09*. New York, NY, USA: ACM, 2009, pp. 169–180.
54. Xu W., Ma K., Trappe W., Zhang Y., “Jamming sensor networks: attack and defense strategies,” *IEEE Network*, vol. 20, no. 3, pp. 41–47, May–June 2006.
55. Xu W., Wood T., Trappe W., Zhang Y., “Channel surfing and spatial retreats: defenses against wireless denial of service,” in *Proceedings of the 3rd ACM workshop on Wireless security WiSe ’04*. New York, NY, USA: ACM, 2004, pp. 80–89.
56. Xu W., Trappe W., Zhang Y., Wood T., “The feasibility of launching and detecting jamming attacks in wireless networks,” in *Proceedings of the 6th ACM International symposium on Mobile ad hoc networking and computing MobiHoc ’05*. New York, NY, USA: ACM, 2005, pp. 46–57.
57. Pickholtz R.L., Schilling D.L., Milstein L.B., “Theory of Spread-Spectrum Communications - A Tutorial,” *IEEE Transactions on Communications*, vol. 30, no. 5, pp. 855–884, May 1982.
58. Anderson R.J. , Security Engineering: A Guide to Building Dependable Distributed Systems. New York, NY, USA: John Wiley & Sons, Inc., 2001.
59. Woo A., Tong T., Culler D., “Taming the underlying challenges of reliable multihop routing in sensor networks,” in *Proceedings of the 1st International conference on Embedded networked sensor systems SenSys ’03*. New York, NY, USA: ACM, 2003, pp. 14–27.
60. Proano A., Lazos L., “Selective jamming attacks in wireless networks,” in *2010 IEEE International Conference on Communications*, May 2010, pp. 1–6.
61. Law Y.W., Hartel P., Den Hartog J., Havinga P., “Link-layer jamming attacks on S-MAC,” in *Proceedings of the Second European Workshop on Wireless Sensor Networks*, January–February 2005, pp. 217–225.
62. O’Flynn C.P., “Message Denial and Alteration on IEEE 802.15.4 Low-Power Radio Networks,” in *2011 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Feb 2011, pp. 1–5.
63. Wood A.D., Stankovic J.A., Zhou G., “DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks,” in *4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, Jun 2007, pp. 60–69.
64. Menezes A.J., Van Oorschot P.C., Vanstone S.A., *Handbook of Applied Cryptography*. CRC Press, 2001.
65. CONET, “Cooperating objects network of excellence, european commission, 7th framework programme, grant agreement n. 224053,” http://www.cooperating-objects.eu/, 2008.
66. PLANET, “Platform for the deployment and operation of heterogeneous networked cooperating objects, european commission, 7th framework programme, grant agreement n. 257649,” http://www.planet-ict.eu/, 2010.
67. Buttyan L., Gessner D., Hessler A., Langendoerfer P., “Application of wireless sensor networks in critical infrastructure protection: challenges and design options [security and privacy in emerging wireless networks],” *IEEE Wireless Communications*, vol. 17, no. 5, pp. 44–49, Oct 2010.

68. Albano M., Chessa S., Di Pietro R., "Information assurance in critical infrastructures via wireless sensor networks," in *Proceedings of the 4th International Conference on Information Assurance and Security ISIAS '08*, 2008, pp. 305–310.
69. Crossbow Technology Inc., "MPR-MIB Users Manual," http://bullseye.xbow.com:81/Support/, Jun 2007. [Online]. Available: http://bullseye.xbow.com:81/Support/
70. Zoelertia, "Zoelertia Z1 Datasheet," http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf, Mar 2010. [Online]. Available: http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf
71. Viega J., Evans D., "Separation of concerns for security," in *Proceedings of the ICSE Workshop on Multidimensional Separation of Concerns in Software Engineering*, 2000, pp. 126–129.
72. Daidone R., Dini G., Tiloca M., "STaR source code," Nov 2012.
73. Wong C.K., Gouda M., Lam S.S., "Secure group communications using key graphs," *IEEE_J_NET*, vol. 8, no. 1, pp. 16–30, Feb 2000.
74. Dini G., Tiloca M., "Considerations on security in zigbee networks," in *Proceedings of the 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing SUTC '11*, Jun 2010, pp. 58–65.
75. Maerien J., Michiels S., Huygens C., Joosen W., "MASY: MAnagement of Secret keYs for federated mobile wireless sensor networks," in *Proceedings of the 6th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Oct 2010, pp. 121–128.
76. Zhong S., Chuang L., Fengyuan R., Yixin J., Xiaowen C., "An efficient scheme for secure communication in large-scale wireless sensor networks," in *Proceedings of the WRI International Conference on Communications and Mobile Computing CMC '09*, vol. 3, Jan 2009, pp. 333–337.
77. Garcia-Morchon O., Baldus H., "The ANGEL WSN Security Architecture," in *Proceedings of the 2009 3rd International Conference on Sensor Technologies and Applications SENSORCOMM '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 430–435.
78. Gu W., Dutta N., Chellappan S., Xiaole B., "Providing end-to-end secure communications in wireless sensor networks," *IEEE Transactions on Network and Service Management*, vol. 8, no. 3, pp. 205–218, Sep 2011.
79. Hyun S., Ning P., Liu A., Du W., "Seluge: Secure and dos-resistant code dissemination in wireless sensor networks," in *Proceedings of the 2008 International Conference on Information Processing in Sensor Networks*, Apr 2008, pp. 445–456.
80. Lanigan P.E., Gandhi R., Narasimhan P., "Sluice: Secure dissemination of code updates in sensor networks," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, Jul 2006, pp. 53–62.
81. Dini G., Savino, I.M., "A Security Architecture for Reconfigurable Networked Embedded Systems," *International Journal of Wireless Information Networks*, vol. 17, pp. 11–25, 2010.
82. Matthys N., Huygens C., Hughes D., Michiels S., Joosen W., "A component and policy-based approach for efficient sensor network reconfiguration," in *Proceedings of the 2012 IEEE International Symposium on Policies for Distributed Systems and Networks*, Jul 2012, pp. 53–60.
83. Liu Z., Peng D., "A security-supportive middleware architecture for pervasive computing," in *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006, pp. 137–144.
84. Daidone R., Dini G., Tiloca M., "STaR: Security Transparency and Reconfigurability for Wireless Sensor Networks programming," in *Proceedings of the 2nd International Conference on Sensor Networks SENSORNETS '13*, Feb 2013.
85. U.S. National Security Agency (NSA), "Skipjack and kea algorithm specifications," May 1998. [Online]. Available: http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf

86. Eastlake D., Jones P., "Rfc 3174 - us secure hash algorithm 1 (sha1)," http://tools.ietf.org/html/rfc3174, Sep 2001. [Online]. Available: http://tools.ietf.org/html/rfc3174
87. De Dios J.R.M., Lferd K., De San Bernabé A., Núñez G., Torres-González A., Ollero, A., "Cooperation Between UAS and Wireless Sensor Networks for Efficient Data Collection in Large Environments," *Journal of Intelligent and Robotic Systems*, vol. 70, no. 1-4, pp. 491–508, 2013.
88. "nesC: A Programming Language for Deeply Networked Systems." [Online]. Available: http://nescc.sourceforge.net/

Acknowledgment

For my Ph.D. dissertation and for the time I spent as a Ph.D student I would like to thank:

- *My advisor, Prof. Gianluca Dini* for having taught me to approach research with method and passion, for having encouraged me to meet high targets, improving my skills from both human and professional standpoint.
- *My family* for having supported me during these years and for believing in me.
- *Prof. Giuseppe Anastasi*, for his precious support during the work we have made in cooperation.
- *Ofelia, Selene and Alessandro*, because just a few are friends for real, because friends do not judge you, because they are close to you in spite of distance.
- *Dario*, because his irony keeps both my feet on the ground.
- *Gabriela M.*, for sharing not only a flat, but the energy to cope with daily difficulties.
- *The H_2Ogym's angels*. Gabriela P., Lucia, Rossella and Viola, for our good time, before, during and after trainings, and for the friendship that derived from this.
- Pietro and Ilaria, because their trainings not only improved my physical and mental health, but also lightened some difficult periods.
- *People I have worked with in the PLANET project*.
- *People I have worked with in the CONET network of excellence*.
- *Colleagues that have been present always or just for a while, that are near or far away*. Pericle Perazzo, Nilo Redini, Giacomo Tanganelli, Daniel Cesarini, Valerio Luconi, Giovanni Nardini, Angelica Lo Duca, Francesco Giurlanda, Marco Tiloca, Andrea Saracino, Alessandro Pischedda, Domenico De Guglielmo, Luca Cassano, Stefano Abate, Stefano Campanelli, Carlo Vallati, Antonio Virdis, Maria Antonietta La Polla, Chiara Orsini, Fabio Pezzoni, Valerio Arnaboldi, Lorenzo Valerio, Davide Di Baccio, Simone Martini, Adriano Fagiolini, Adriano Faggiani, Alessandro Improta, Giovanni Stea, Mario Cimino, Eleonora D'Andrea, Hakjeon Bang, Sena Efsun Cebeci, for having made the Ph.D. "journey" happier.
- *Swimming*, because without that kind of experiences I would not have deeply known sacrifice, determination, victory, defeat, and, above all, passion.

Ringraziamenti

Per questa Tesi di Dottorato e per il tempo trascorso insieme durante il percorso vorrei ringraziare:

- *Il mio advisor. Il Prof. Gianluca Dini*, per avermi trasmesso passione e metodo nel fare ricerca e per avermi sottoposto sfide sempre nuove, così da ampliare le mie capacità sia dal punto di vista scientifico che umano.
- *La mia famiglia*, per avermi supportata e per aver creduto in me, sempre.
- *Ofelia, Selene e Alessandro*, perché i veri amici sono pochi, perché non ti giudicano, perché sanno essere vicini a dispetto della distanza.
- *Dario*, perché la sua ironia riesce a tenermi coi piedi saldamente per terra.
- *Gabriela M.*, per aver condiviso non solo un appartamento, ma anche l'energia per affrontare le difficoltà quotidiane.
- *Le H_2Ogym's angels*. Gabriela P., Lucia, Rossella e Viola, per le risate prima, durante e dopo le lezioni in piscina e per la bella amicizia che ne è nata.
- Pietro e Ilaria, perché non solo hanno migliorato la mia salute sia fisica che mentale con i loro allenamenti, ma hanno anche alleggerito i momenti pesanti con la loro simpatia.
- *Le persone con cui ho lavorato per il progetto PLANET*.
- *Le persone con cui ho lavorato per la rete d'eccellenza CONET*.
- *Colleghi stabili, di passaggio, vicini e lontani*. Pericle Perazzo, Nilo Redini, Giacomo Tanganelli, Daniel Cesarini, Valerio Luconi, Giovanni Nardini, Angelica Lo Duca, Francesco Giurlanda, Marco Tiloca, Andrea Saracino, Alessandro Pischedda, Domenico De Guglielmo, Luca Cassano, Stefano Abate, Stefano Campanelli, Carlo Vallati, Antonio Virdis, Maria Antonietta La Polla, Chiara Orsini, Fabio Pezzoni, Valerio Arnaboldi, Lorenzo Valerio, Davide Di Baccio, Simone Martini, Adriano Fagiolini, Adriano Faggiani, Alessandro Improta, Giovanni Stea, Mario Cimino, Eleonora D'Andrea, Hakjeon Bang, Sena Efsun Cebeci, per aver reso più felice il "viaggio" da allieva di Dottorato.
- *Il nuoto*, perché senza determinate esperienze non avrei conosciuto così profondamente sacrificio, determinazione, vittoria, sconfitta e, soprattutto, passione.