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

Solutions and Tools for Secure Communication in Wireless Sensor Networks

Autore:

Marco Tiloca _____

Relatore:

Prof. Gianluca Dini _____

Anno 2013
SSD ING-INF/05

To Euterpe,
for having kept me alive
in my darkest hours.

Sommario

La comunicazione sicura è considerata un requisito fondamentale nelle applicazioni basate su *Wireless Sensor Networks* (*WSNs*). Essa comprende differenti aspetti, tra cui confidenzialità, integrità e autenticità delle informazioni trasmesse, una accurata gestione del materiale crittografico, e una efficace prevenzione e reazione contro eventuali attacchi. Tuttavia, le WSNs sono principalmente composte da dispositivi dalle risorse limitate, specialmente in termini di memoria disponibile, capacità computazionale, velocità trasmissiva, e disponibilità energetica.

Garantire la comunicazione sicura nelle WSNs risulta quindi essere più difficile che in altri tipi di rete. Infatti, diventa ancora più importante raggiungere un compromesso tra l'efficacia e l'efficienza delle soluzioni adottate. Per di più, specifiche classi di dispositivi o tecnologie possono richiedere l'adozione di soluzioni ad-hoc. Inoltre, è necessario gestire efficientemente il materiale crittografico, e adattarsi dinamicamente a cambiamenti dei requisiti di sicurezza. Infine, possibili attacchi e relative contromisure devono essere vagliati attentamente sin dalla progettazione della rete.

Questa tesi di dottorato considera la comunicazione sicura nelle WSNs, e consiste nei seguenti contributi. Per prima cosa, valutiamo le performance dei servizi di sicurezza offerti dallo standard *IEEE 802.15.4*. Quindi, consideriamo la tecnologia *ZigBee*, ne analizziamo i servizi di sicurezza, e proponiamo possibili soluzioni ad alcuni difetti e inefficienze. Successivamente, presentiamo *HISS*, uno schema per la gestione delle chiavi crittografiche altamente scalabile ed efficiente, in grado di contrastare gli attacchi di collusione a fronte di un graduale calo delle prestazioni. Quindi, presentiamo *STaR*, una componente software per WSNs, atta a proteggere simultaneamente più flussi di traffico. Oltre che trasparente all'applicazione, è riconfigurabile dinamicamente, e quindi adatta a possibili cambiamenti dei requisiti di sicurezza. Infine, descriviamo *ASF*, il nostro framework per la simulazione di attacchi su WSNs. ASF aiuta il progettista a valutare quantitativamente gli effetti di possibili attacchi, a classificarli in base alla loro gravità, e a selezionare le contromisure più appropriate.

Abstract

Secure communication is considered a vital requirement in *Wireless Sensor Network* (*WSN*) applications. Such a requirement embraces different aspects, including confidentiality, integrity and authenticity of exchanged information, proper management of security material, and effective prevention and reaction against security threats and attacks. However, WSNs are mainly composed of resource-constrained devices. That is, network nodes feature reduced capabilities, especially in terms of memory storage, computing power, transmission rate, and energy availability.

As a consequence, assuring secure communication in WSNs results to be more difficult than in other kinds of network. In fact, trading effectiveness of adopted solutions with their efficiency becomes far more important. In addition, specific device classes or technologies may require to design ad-hoc security solutions. Also, it is necessary to efficiently manage security material, and dynamically cope with changes of security requirements. Finally, security threats and countermeasures have to be carefully considered since from the network design phase.

This Ph.D. dissertion considers secure communication in WSNs, and provides the following contributions. First, we provide a performance evaluation of *IEEE 802.15.4* security services. Then, we focus on the *ZigBee* technology and its security services, and propose possible solutions to some deficiencies and inefficiencies. Second, we present *HISS*, a highly scalable and efficient key management scheme, able to contrast collusion attacks while displaying a graceful degradation of performance. Third, we present *STaR*, a software component for WSNs that secures multiple traffic flows at the same time. It is transparent to the application, and provides runtime reconfigurability, thus coping with dynamic changes of security requirements. Finally, we describe *ASF*, our attack simulation framework for WSNs. Such a tool helps network designers to quantitatively evaluate effects of security attacks, produce an attack ranking based on their severity, and thus select the most appropriate countermeasures.

Contents

List of Figures

List of Tables

1

Introduction

In the recent years, *Wireless Sensor Networks* (*WSNs*) have become a popular and widely adopted technology, both in the academic and industrial world. They can be profitably used in several fields of application, including military surveillance, home automation, and environment monitoring. Success gained by WSNs is mostly due to the availability of standard technologies and protocols, low cost of sensor devices, ease of use, as well as an active worldwide community support.

However, WSNs are typically deployed in unattended, unsecure, or even hostile environments. Thus, they can be easily affected by a number of *security attacks* [49][89][98]. Specifically, *logical attacks* aimed at thwarting communication can be easily performed [17][24][59][103][106]. For instance, a determined adversary can intercept network messages, in order to alter, replicate, or even discard them, as well as inject fake ones. Also, unsecure environments ease an adversary to perform *physical attacks* [8][81], i.e. to physically attack sensor devices in order to reprogram, misplace, or break them. In the end, this most likely determines unreliable data collection, and may even results in safety issues.

It follows that it is vital to protect WSNs by adopting appropriate security countermeasures. However, with respect to other network technologies, properly securing WSNs results to be a more difficult task. This is mainly due to the scarce amount of resources available on sensor devices, in terms of memory capacity and computation capability [84][85]. Also, sensor nodes are typically battery powered, thus limiting power consumption is a key aspect in WSN applications [39]. Therefore, we argue that providing security in WSNs should be based on the following recommendations.

1. Adopted solutions must be both effective and *efficient*. That is, they must result in an affordable impact on performance, and be scalable with the network size.
2. Security countermeasures must *accommodate* different traffic flows, and be *adaptable* to dynamic changes in network conditions.
3. In order to select adequate countermeasures, it must be possible to evaluate *impact* and *effects* of security attacks since from the network design phase.

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 describe security services provided by the *IEEE 802.15.4* standard [52], and present our implementation of its security sublayer for the *TinyOS* platform [100] and the *TmoteSky* motes [74]. We relied on our implementation to experimentally evaluate how IEEE 802.15.4 security services impact on memory occupancy, network performance, and energy consumption [85]. In particular, we show that IEEE 802.15.4 security services require an affordable amount of memory, and have a meaningful impact on network performance and energy consumption. We believe this is an important step towards a quantitative analysis that allows designers and implementers to properly define a security-performance trade-off.

Chapter 3 considers the security model provided by the *ZigBee* technology [116] and its *Smart Energy* application profile [117]. Although it provides a good trade-off between security and complexity, this model presents deficiencies concerning key and certificate management that may limit its application [45]. Specifically, we highlight a deficiency in cryptographic key management, and propose to manage rekeying at the application level. Such a solution makes it possible to select the rekeying scheme that better suits the application requirements and constraints. Also, we show that the objectives of openness and interoperability may result in a not scalable certificate management, due to the limited storage resources of ZigBee devices. In order to overcome this problem, we propose a home-certification mechanism that drastically reduces the storage requirements without endangering security.

In Chapter 4, we present *HISS*, a highly scalable scheme for *group rekeying* based on logical subgrouping [47]. HISS has the following merits. First, it allows for securely and efficiently performing group rekeying upon a user join or leave, thus assuring both backward and forward security. In order to do that, it requires a number of rekeying messages that is small, constant, and independent of the group size. Second, in terms of memory occupancy, HISS requires every user to store $\mathcal{O}(\sqrt{n})$ secrets, which is an affordable storage overhead in most practical cases, even encompassing resource-scarce devices such as sensor nodes. Third, HISS provides a recovery protocol aimed at restoring group security upon a *collusion attack*. The recovery protocol does not require to re-initialise the group, it is affordable on customary platforms, and displays a form of graceful degradation. That is, the protocol communication performance degrades with the number of compromised subgroups. Finally, HISS makes it possible to define policies of allocation of users to subgroups that practically constrast or even prevent successful instances of collusion attacks.

Chapter 5 presents *STaR*, our security software component for WSNs that protects multiple traffic flows at the same time, according to different security policies [87]. STaR is transparent to the application, which can rely on the same communication interface already in use. Also, it allows users to change security policies and their association to traffic flows at runtime. Finally, we consider our preliminary implementation of STaR for TmoteSky motes, and provide a performance evaluation in terms

of memory occupancy, communication overhead, and energy consumption. Our results show that STaR 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 STaR.

Finally, in Chapter 6, we present *ASF*, our framework for simulative evaluation of attacks in WSNs [46]. ASF provides an *Attack Specification Language* to describe different kinds of attacks, and an *Attack Simulator* to quantitatively evaluate their effects. This allows users to evaluate attacks severity, and thus define protection priorities and select appropriate countermeasures. We consider a realistic application scenario, and use our ASF prototype for the *Castalia* simulator [2] to evaluate the effects of four different attacks. Our results show how considered attacks affect the application and network behavior, and suggest to ensure reliability of communications and provide physical protection to sensor nodes.

2

Security in IEEE 802.15.4 networks

IEEE 802.15.4 is an emerging technology addressing the needs of 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 [52]. IEEE 802.15.4 is adopted in a wide range of application scenarios including environmental monitoring, health-care, military surveillance, and industrial automation. Most of these applications require forms of secure communication including confidentiality, authenticity, and ready detection of replay-attacks. For this reason, IEEE 802.15.4 specification includes a number of security provisions and options.

Security and performance of IEEE 802.15.4 have been extensively analysed, although separately. Relevant works include [76][102] as to security analysis, as well as [60][62] as to performance analysis. In contrast, a thorough analysis of the impact that security provisions and options have on IEEE 802.15.4 performance is missing. Some related works have been presented, but they face either with specific aspects, such as cipher design [110], or with collateral although important issues, such as key management [37][58].

What is really missing is an analysis providing quantitative indications regarding consumption of system resources due to security services provided by IEEE 802.15.4. We believe that this analysis is crucial. Security and performance compete for the same system resources, namely memory, CPU, bandwidth and energy, that are scarce in low power, low cost sensor devices. Therefore, quantitative indications regarding resources consuption are fundamental to design and implement adequate performance-security trade-offs in IEEE 802.15.4-based applications.

For these reasons, we have performed an experimental evaluation of impact and costs of IEEE 802.15.4 security services [85]. Specifically, we referred to a free implementation of the IEEE 802.15.4 specification for TinyOS on TmoteSky motes [4]. We have extended it with an implementation of the IEEE 802.15.4 security sublayer, which is compliant to the standard specification [53]. To the best of our knowledge, this is the first available free implementation of IEEE 802.15.4 including security services.

Our experimental evaluation focuses on three performance aspects, namely memory occupancy, network performance, and energy consumption, and has a twofold objective. On the one hand, we aim at evaluating how security impacts on these three aspects. That is, we are interested in determining how security services (e.g. confidentiality and/or authenticity) and security options (e.g. message integrity code length) influence such aspects. On the other hand, we are willing to devise a model that allows designers and implementers to carry out, for example at pre-deployment, simulative and/or analytical performance analysis that include security too.

In this chapter, we report results of our evaluation activity. As to *memory occupancy*, we show that security requires a non negligible although affordable amount of memory. This result is relevant as WSNs often comprise devices whose storage capabilities are severely limited. For instance, TmoteSky motes have 48 Kbytes of available memory and our implementation of the IEEE 802.15.4 security sublayer requires just the 9.4% of that memory on the sender side, and the 12.9% on the receiver side. As to *network performance*, we show that the security impact derives from the fact that, in the most general case, secured frames are larger than unsecured ones (frame expansion) and that securing frames requires additional frame processing (extra processing). We show that securing communications reduces the amount of transmitted data frames of up to 33.8%. Finally, frame expansion and extra processing influence also *energy consumption*, which is one of the main issues in WSNs. We show that the major impact is due to the transmission of expanded data frames, which represents the 61.12% of the overall extra energy consumption in the presence of security.

2.1 IEEE 802.15.4 security services

Two different kinds of device can participate in an IEEE 802.15.4 network: *Full-Function Devices* (*FFDs*) and *Reduced-Function Devices* (*RFDs*). In particular, one FFD is elected as the *Personal Area Network* (*PAN*) Coordinator and is responsible for network and security management. In the following, we consider a *beacon enabled PAN*, that is the PAN Coordinator periodically broadcasts *beacon* frames within the network. Specifically, the MAC attribute *BeaconOrder* defines the interval at which the PAN Coordinator broadcasts beacon frames.

IEEE 802.15.4 provides a number of security services and makes them available to the higher layers. In particular, data confidentiality, data authenticity and replay protection are supported on a per-frame basis. The standard includes a security suite based on the *Advanced Encryption Standard* (*AES*) 128 bits symmetric-key cryptography. The security suite relies on three elements: an *Auxiliary Security Header* (*ASH*), *security modes* and settings, and *security procedures*.

If communications are secured, senders build the ASH, insert it next to the standard MAC header (see Figure 2.1), and secure frames before transmitting them. According to the information carried by the ASH, recipients retrieve the right cryptographic key and correctly unsecure MAC frames. More in detail, the ASH carries in-

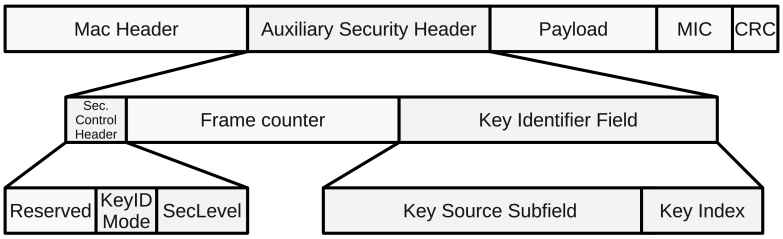


Figure 2.1. Auxiliary Security Header (ASH).

formation required for the security processing, including i) the specified *security mode* and its options, ii) a frame counter value for the anti-replay service, and, finally, iii) the *KeyIdMode* according to which the key retrieval procedure is supposed to take place.

In particular, the KeyIdMode indicates whether the cryptographic key has to be obtained implicitly or explicitly. The IEEE 802.15.4 standard provides four different KeyIdModes, that is four different ways to retrieve cryptographic keys. More specifically, in *KeyIdMode0* the key is determined implicitly from the originator and the recipient(s) of the frame, and the Key Identifier field of the ASH is not present. In *KeyIdMode1* the key is determined from the Index subfield of the Key Identifier field of the ASH, in conjunction with the macDefaultKeySource value, which is predetermined by network devices. In *KeyIdMode2* the key is determined explicitly from the 4 bytes Key Source subfield and the 1 byte Key Index subfield of the Key Identifier field. Finally, in *KeyIdMode3* the key is determined in the same way as in KeyIdMode2, but the Key Source subfield is 8 bytes in size instead of 4.

Security mode	**Data confidentiality**	**Data authenticity**	**MIC size (bytes)**
CTR	ON	OFF	-
CBC_MAC_4	OFF	ON	4
CBC_MAC_8	OFF	ON	8
CBC_MAC_16	OFF	ON	16
CCM_4	ON	ON	4
CCM_8	ON	ON	8
CCM_16	ON	ON	16

Table 2.1. Security modes.

Three different kinds of security modes are available: encryption only (*CTR*), authentication only (*CBC_MAC*), and both encryption and authentication (*CCM*). In particular, CBC_MAC and CCM rely on a *Message Integrity Code* (*MIC*), whose size can be either 4, 8, or 16 bytes. By choosing and properly setting the security mode to be used, it is possible to deal with applications' constraints and security requirements. Table 2.1 provides an overview of the available security modes.

By means of the standard *security procedures*, it is possible to secure and unsecure MAC frames, assure a minimum security level, retrieve cryptographic keys, deal with blacklisted nodes, and verify frames' freshness by means of the *Frame Counter* field of the ASH. Securing/unsecuring operations rely on a fresh *nonce* value, that is a randomly generated number used to prevent replay attacks. Nonces are generated by senders and checked by recipients.

Security material is stored into two different tables on each device, that is a *Key Table* and a *Device Table*. The former contains cryptographic keys and their identifiers, while the latter includes information about other sender devices, such as the highest frame counter value received by each one of them. Security procedures are responsible for accessing and updating data structures, and verifying their consistency.

Finally, IEEE 802.15.4 does not concern about key establishment and devices authentication, which are potentially entrusted to the higher layers. Thus, both senders and recipients have to share common security settings and store the necessary security material before secure communication can actually take place.

2.2 Security sublayer implementation

We extended the open source implementation of IEEE 802.15.4 for the TinyOS platform currently available at [4]. In particular, we implemented IEEE 802.15.4 security services and procedures responsible for protecting MAC data frames, with reference to the TmoteSky motes [74] and the CC2420 chipset [99]. The source code of our implementation can be found at [5].

MAC layer security relies on two main sets of services, namely frame handling and actual security procedures. *Frame handling* has been extended in order to properly manage the auxiliary security header in case data frames require to be protected. *Security procedures* have been implemented on both the sender and the receiver side, according to the guidelines and practices described by the IEEE 802.15.4 standard.

While implementing our security suite, we made reference to the security mechanisms provided by the CC2420 chipset. All security modes described in Section 2.1 are available and can be selected on a per-frame basis, according to the application requirements on the sender side. CC2420 provides cryptographic primitives based on AES 128 bits encryption, and hardware support for the IEEE 802.15.4 security services. MAC frames protection can be performed in two different ways: *stand-alone* or *in-line*. The former encrypts MAC frames into a proper RAM buffer, while the latter secures and unsecures frames within the transmit buffer TXFIFO and the receive buffer RXFIFO, respectively. In the following, we refer to the in-line security operations.

CC2420 determines the security mode to be used according to the SECCTRL0 and SECCTRL1 registers settings. That is, these registers have to be properly set before the actual security operations take place. Besides, before issuing a security command strobe, it is necessary to set the cryptographic key to be used into the

KEY0 or KEY1 register. Finally, in order to detect replay attacks, a nonce is written in the TXNONCE (on the sender side) or RXNONCE (on the recipient side) register.

Outgoing frames protection is accomplished by issuing the STXENC command strobe, which actually secures the frame within the TXFIFO buffer and then transmits it. On the other hand, recipient nodes invoke the SRXDEC command strobe, which unsecures the frame inside RXFIFO and makes it available to the higher layers.

2.3 Performance evaluation

In order to test our security sublayer, we have used two TmoteSky motes, equipped with a 48 Kbytes ROM, and a 8 MHz MSP430 microcontroller with a 10 Kbytes RAM. In particular, we have considered a beacon enabled network with BeaconOrder 7. An FFD acts as the PAN Coordinator and a single RFD acts as sender. Both the RFD and the FFD share a common cryptographic symmetric key. The PAN Coordinator unsecures received protected data frames, and sends ACK frames back. On the other hand, the RFD continuously transmits protected data frames to the PAN Coordinator, and waits for ACK frames back. We consider data frames whose payload is 18 bytes in size. Since the IEEE 802.15.4 standard does not permit to secure ACK frames, they are neither encrypted nor authenticated.

2.3.1 Memory consumption

The amount of available memory on a TmoteSky mote may represent a severe constraint while developing applications or, as in our case, while extending MAC layer capabilities. We have evaluated the memory overhead due to the presence of IEEE 802.15.4 security sublayer by comparing the amount of used memory both in the presence and in the absence of security. In the absence of security, memory is necessary to allocate the application and TinyOS images. In the presence of security, additional memory is necessary to allocate security services and data structures. As to security, we have considered the KeyIdMode2 and the CCM_16 security mode.

Figure 2.2 shows the memory footprints of the PAN Coordinator (columns A, B) and the RFD (columns C, D) with and without security services, respectively. Y-axis reports the absolute memory occupancy in bytes, whereas percentages express memory occupancy as a fraction of the available memory (i.e. 48 Kbytes).

On the PAN Coordinator, memory occupancy without security is 33.12 Kbytes (i.e. 69% of the available memory). It becomes 39.31 Kbytes (81.9%) when security is used. It follows that security causes an increase of memory occupancy of about 6.19 Kbytes (12.9%). This leaves 8.69 Kbytes (18.1%) free for other uses. As to the RFD, memory occupancy without security is 34.43 Kbytes (71.7%), whereas it becomes 38.94 Kbytes (81.1%) with security. It follows that security causes an increment of 4.51 Kbytes (9.4%). This leaves 9.06 Kbytes (18.9%) free for other uses. Without security, the difference in size between the RFD and the PAN Coordinator is merely due to

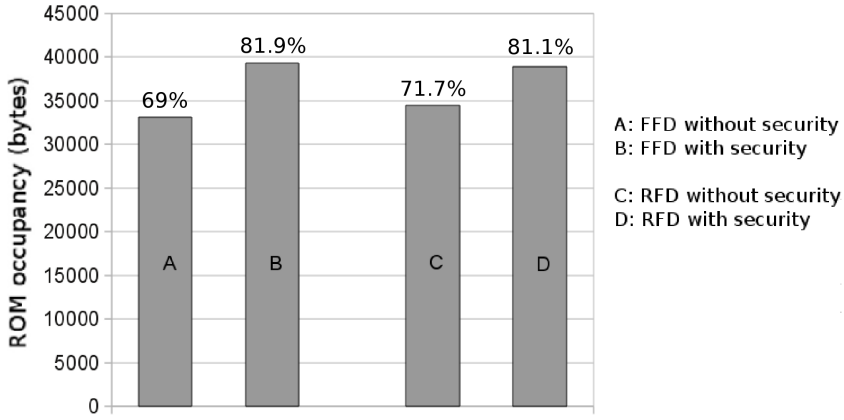


Figure 2.2. Memory consumption.

their different basic operations. In the presence of security, the memory occupancy increases both on the RFD and the PAN Coordinator. However, unlike RFDs, the PAN Coordinator is required to deal with larger security structures in a more complex way.

Notice that the amount of free memory is important because other high level security services might be necessary. For instance, let us consider key establishment. If a designer wishes to use Elliptic Curve Diffie-Hellman (ECDH), a possibility is the implementation in the TinyECC suite [11]. However, only the unoptimized version could be accommodated in the free memory as it requires 4446 bytes on a TmoteSky mote. Of course, a more memory-efficient implementation of IEEE 802.15.4 security sublayer could save more memory. However, although we believe that our implementation is affordable given the limited memory overhead it causes, we claim that our aim is not to achieve a highly optimized implementation but, rather, to provide an early experimental framework to start quantitative evaluations.

Also, we have considered the impact of data structures and security operations, separately. Figure 2.3 shows the memory usage breakdowns on the PAN Coordinator and the RFD side, respectively. In particular, the PAN Coordinator requires 1228 bytes (2.6%) for security services and 4958 bytes (10.3%) for security data structures and their relative operations. The RFD requires 826 bytes (1.7%) for security services and 3688 bytes (7.7%) for security data structures and their relative operations.

It is important to notice that memory consumption of data structures varies with the number of RFDs and cryptographic keys, whereas memory consumption of the code does not. Intuitively, the Device Table size grows with the number of RFDs, whereas the Key Table grows with the number of keys. However, the number of RFDs and the number of keys may depend on the design choices. For instance, one possible choice is that all devices in the network share a single “network key”. An alternative choice is that each RFD may share a private secret key with the PAN Coordinator.

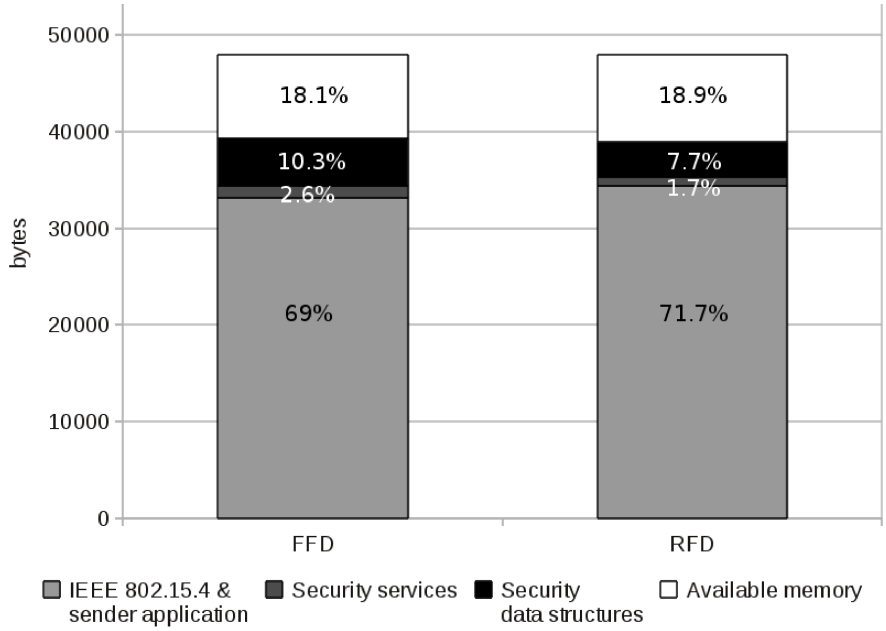


Figure 2.3. Memory occupancy breakdown.

2.3.2 Impact on network performance

In this section, we discuss the impact of security on network performance by evaluating the decrement of transmitted data frames due to the presence of security. Operatively, we consider a single RFD continuously transmitting data frames to the PAN Coordinator for a given amount of time T, as described at the beginning of Section 2.3. Then, we count the number of both secured and unsecured frames transmitted during such an amount of time. Finally, we perform the subtraction of the latter number of frames from the former. As to secured communication, we consider key retrieval mode KeyIdMode2 and all security modes. Provided results are averaged over ten repetitions lasting $T = 120$ s each. In order to count data frames without introducing any additional delay, we used a Texas Instruments IEEE 802.15.4/Zigbee packet sniffer equipped with a CC2430 chipset [6].

Table 2.2 shows the number of transmitted data frames in the different security modes. The column “Mode” lists security modes, and “No security” specifies that frames are not secured. The parameter F refers to the number of data frames transmitted when security is not used, whereas F_e, $F_{a_x}(x = 4, 8, 16)$ and $F_{ea_x}(x = 4, 8, 16)$ specify the number of data frames transmitted when encryption (e), authentication (a) or both (ea) are used, respectively. When authentication is used, x specifies the MIC size. The column “Frame size” specifies the size of the frame for each security mode. Finally, the column “Decrement” specifies the transmitted data frame decrement for each security mode, with respect to the “No security” mode. As it

Mode	Parameter	# frames	Decrement (%)	Frame size (bytes)
No security	F	7685.5	-	27
CTR	F_e	5671.6	26.2	37
CBC_MAC_4	F_{a_4}	5534.8	28	41
CBC_MAC_8	F_{a_8}	5371.6	30.1	45
CBC_MAC_16	$F_{a_{16}}$	5110.3	33.5	53
CCM_4	F_{ea_4}	5514.6	28.2	41
CCM_8	F_{ea_8}	5374.4	30.1	45
CCM_16	$F_{ea_{16}}$	5082.1	33.9	53

Table 2.2. Number of transmitted frames vs. security modes.

turns out, security causes quite a tangible transmitted data frame decrement, ranging from 26.2% in the CTR mode, up to 33.9% in the CCM_16 mode.

The overhead introduced by security services consists in two elements, the *communication overhead* C and the *processing overhead* P. The communication overhead C is due to the extra bytes that it is necessary to transmit in the presence of security. These extra bytes account for the additional ASH and the MIC field. The processing overhead P is due to the extra processing necessary to parse the ASH, compute the MIC, and secure data frames. The objective of our analysis is to separate the contributions of communication and processing from the total overhead. We have accomplished such a task in the case of CCM_16, which causes the largest overhead (see Table 2.2) from both the processing (encyphering and hashing) and the communication (largest MIC) viewpoint.

The communication overhead C has been evaluated as the difference between F, the number of transmitted data frames in the "No security" mode, and $F^*_{a_{16}}$, the number of data frames transmitted when frames have the same size as frames in CBC_MAC_16 mode (see Table 2.2). That is, $C = (F - F^*_{a_{16}})$. In order to evaluate $F^*_{a_{16}}$, we have transmitted unsecured data frames whose payload (18 bytes) has been increased by the size of the MIC field (16 bytes) and the ASH (10 bytes), for a total of 44 bytes. $F^*_{a_{16}}$ amounts to 6091.7 data frames. Frames are pre-prepared in order to avoid any processing overhead. The communication overhead C results in a decrement of transmitted frames equal to 1593.8. As CCM_16 causes a total decrement equal to $(7685.5 - 5082.1) = 2603.4$ frames (see Table 2.2), then the communication overhead represents the 61.2% of the overall overhead (i.e. 2603.4 data frames).

The processing overhead P can be determined as the difference between the total overhead and the communication overhead, i.e. $P = (F - F_{ea_{16}} - C)$. P results equal to 1009.6 unsent data frames, which amounts to the 38.8% of the whole overhead. Note that P is given by three processing overhead subcomponents, i.e. i) building and handling the ASH, ii) computing the MIC, and, finally, iii) encrypting the frame.

2.3.3 Impact on energy consumption

While evaluating how security impacts on WSNs, it is also important to consider the costs in terms of energy consumption due to the additional communications and computations required by any given security mode. More in detail, a security mode requires the following operations: i) extra communications for the transmission of the ASH and the MIC; ii) extra computation at the hardware level for encryption/decryption and/or generation/verification of the MIC; iii) extra computation for security management, e.g. retrieving keys from the Key Table or ASH parsing. Operations i) and ii) are performed by the CC2420 chipset, while operation iii) involves the MSP430 microcontroller. Thus, the extra energy consumption E of a given security mode is given by $E = E_c + E_s + E_p$, where E_c, E_s, and E_p are the energy consumptions associated to i), ii), and iii), respectively.

Each component $E_x, x \in \{c, s, p\}$, is evaluated as $E_x = V_x \times I_x \times t_x$, where V_x and I_x are, respectively, the supply voltage and the absorbed current of the hardware device performing operation x, and t_x is the duration of the operation. The time interval t_x has been computed according to the same experimental method described in Section 2.3.2, by properly bypassing operations not involved in the energy component under exam. Values of V_x and I_x are those specified by the CC2420 and TmoteSky motes data sheets [74][99]. If security is on, we refer to KeyIdMode2 and the CCM_16 security mode, that is the security mode which causes the largest overhead.

E	V	I	t	Involved component
$E_c = 240.54\ \mu J$	$3.6\ V$	$17.4\ mA$	$3.84\ ms$	CC2420
$E_s = 150.34\ \mu J$	$3.6\ V$	$17.4\ mA$	$2.40\ ms$	CC2420
$E_p = 2.66\ \mu J$	$3\ V$	$600\ \mu A$	$1.48\ ms$	MSP430

Table 2.3. Energy consumption overview.

Table 2.3 provides the energy consumption components, and reports related supply voltage, current consumption, and time intervals. More details follow.

- $E_c = V_c \times I_c \times t_c$ is the additional energy comsumed to transmit one secured data frame because of the presence of the ASH and the MIC. Since frames transmission involves the CC2420 chipset, the supply voltage $V_c = 3.6\ V$ and the current consumption $I_c = 17.4\ mA$ have been considered. The time t_c has been calculated as $t_c = t_{c_{ea_{16}}} - t_{c_0}$. In particular, $t_{c_{ea_{16}}}$ is the time required to transmit an unsecured data frame with a 44 bytes payload, thus simulating the presence of the ASH and the MIC. On the other hand, t_{c_0} is the time required to transmit an unsecured basic data frame with a 18 bytes payload.
- $E_s = V_s \times I_s \times t_s$ is the additional energy comsumed to perform hardware encryption and authentication. Since these operations involve the CC2420 chipset,

the supply voltage $V_s = 3.6\ V$ and the current consumption $I_s = 17.4\ mA$ have been considered. The time t_s has been calculated as $t_s = t_{s_{ea_{16}}} - t_{s_0}$. In particular, $t_{s_{ea_{16}}}$ is the time required to transmit a secured data frame having a 18 bytes payload, performing all hardware security operations but not the security management operations. On the other hand, t_{s_0} is the time required to transmit an unsecured basic data frame with a 18 bytes payload.

- $E_p = V_p \times I_p \times t_p$ is the additional energy comsumed to perform security management operations. Since these operations involve the MSP430 microcontroller, the supply voltage $V_p = 3\ V$ and the current consumption $I_p = 600\ \mu A$ have been considered. The time t_p has been calculated as $t_p = t_{p_{ea_{16}}} - t_{p_0}$. In particular, $t_{p_{ea_{16}}}$ is the time required to transmit a data frame having a 18 bytes payload, performing all security management operations but not the actual hardware security operations. Instead, t_{p_0} is the time required to transmit an unsecured data frame with a 44 bytes payload, thus simulating the presence of the ASH and the MIC.

Finally, the overall extra energy consumption due to the presence of security services is $E = E_c + E_s + E_p = 393.54\ \mu J$ per data frame.

3

Security in ZigBee networks

ZigBee is an emerging standard for low-power, low-rate wireless communication which aims at interoperability and encompasses a full range of devices even including low-end battery-powered sensor nodes [116]. It is built upon the physical layer and medium access control defined in the IEEE 802.15.4 standard (2003 version) [51].

ZigBee Specification includes a number of security provisions and options. In particular, ZigBee provides facilities for carrying out secure communication, protecting establishment and transport of cryptographic keys, cyphering frames and controlling devices. ZigBee improves the basic security framework defined in IEEE 802.15.4, focusing also on establishment and distribution of cryptographic keys.

ZigBee Specification provides two security models, namely *Standard Security Mode* and *High Security Mode*. While the former is designed for lower security residential applications, the latter is intended to be used for high security commercial applications. Also, the security model provided by the *Smart Energy Profile* [117] is asserting itself as a reference security model for ZigBee applications, since it constitutes a trade-off between the two standard modes.

In this chapter, we introduce ZigBee security mechanisms and the Smart Energy Profile security model. Then, we show that two critical issues related to cryptographic key renewal and device authentication have not been adequately addressed, and propose our solutions to them. More detailed information can be found in [45][95].

3.1 Overview

ZigBee is a specification for a suite of high level communication protocols, intended for devices equipped with small and low-power digital radios based on the IEEE 802.15.4 standard [51]. As reported in [35], ZigBee and IEEE 802.15.4 are standard-based protocols which provide the network infrastructure required for wireless sensor network applications. As depicted in Figure 3.1, IEEE 802.15.4 defines the physical and MAC layers, while ZigBee defines the network and application layers.

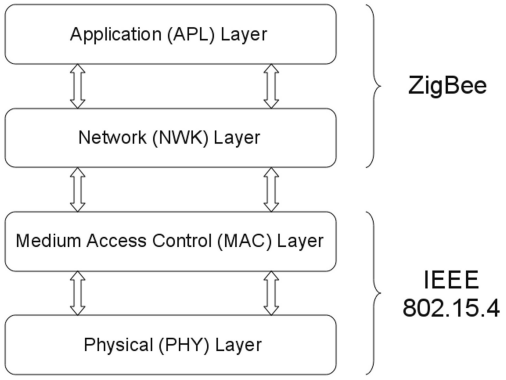


Figure 3.1. ZigBee protocol stack.

The IEEE 802.15.4 MAC layer provides reliable communication between a node and its immediate neighbors, addressing collision avoidance and improving efficiency. The MAC layer also assembles and decomposes data packets and frames, while the physical layer provides the interface to the physical transmission medium (e.g. radio).

ZigBee places itself on top of the IEEE 802.15.4 PHY and MAC layers. Basically, it is formed by the application (APL) layer and the network (NWK) layer. Among other things, the application layer specifies frame formats for transporting data and provides a data service to the applications, while the network layer handles network management and routing by invoking actions in the MAC layer. Security is provided in a cross-layered fashion, involving both the application and the network layer.

According to the Specification [116], a ZigBee network may comprise three types of devices: *Coordinator*, *Router*, and *end device*. With reference to the device types in an IEEE 802.15.4 network, the ZigBee Coordinator corresponds to the PAN Coordinator, a Router corresponds to a Coordinator and an end device corresponds to an RFD or an FFD which is neither a Coordinator nor the PAN Coordinator. In the following, we use the ZigBee terminology to indicate devices.

The ZigBee network layer (NWK) supports *Star*, *Tree*, and *Mesh* topologies. In the Star topology, the network is controlled by the Coordinator, which is responsible for initiating and maintaining the devices on the network, while end devices directly communicate with the Coordinator. In Mesh and Tree topologies, the Coordinator is responsible for starting the network and for choosing certain key network parameters. However, the network may also be extended through the use of ZigBee Routers, while routes are established by means of a routing protocol similar to the Ad hoc On-demand Distance Vector (AODV) protocol [27]. In Tree networks, Routers move data and control messages through the network using a hierarchical routing strategy.

3.2 ZigBee application profiles

ZigBee provides some *application profiles* that specify possible collections of devices, and a set of messages used by devices to communicate with one another. Each application profile describes also some *clusters*, i.e. sets of parameters and commands (some mandatory) that devices have to use in order to interoperate within the network. Nowadays, the most important and promising ZigBee application profiles are *Home Automation* [115] and *Smart Energy* [117].

The ZigBee Home Automation profile defines device descriptions and standard practices for applications needed in a residential or light commercial environment. In particular, it provides standard interfaces and device definitions to allow interoperability among ZigBee devices produced by various manufacturers of home automation products. Installation scenarios range from a single room to an entire home. The key application domains included so far are lighting, HVAC, window shades and security. Also, this profile primarily focuses on sporadic real time control of devices. Other applications will be added in future versions.

The ZigBee Smart Energy Profile (SEP) provides device descriptions and standard practices for Demand Response and Load Management Smart Energy applications, and is tailored for residential or light commercial environment. Possible scenarios include single homes or even an entire apartment complex. Currently, key application domains are metering, pricing, and demand response and load control applications. SEP specification provides standard interfaces and device definitions to allow interoperability among ZigBee devices produced by various manufacturers of electrical equipment, meters, and Smart Energy enabling products.

3.3 ZigBee security services

ZigBee security mechanisms fit very well with the security services provided by the IEEE 802.15.4 MAC layer. Communication can be protected by means of symmetric key encryption, in an end-to-end fashion. In particular, a ZigBee network must comprise one *Trust Center*, a node, typically the ZigBee Coordinator, which provides key management and other security services. ZigBee security relies on the *Advanced Encryption Standard* (*AES*) [78] and the *CCM** Mode (i.e. encryption and authentication). Also, two distinct security modes are available, that is the *Standard Security Mode* and the *High Security Mode*.

Security is provided on an end-to-end basis in a cross-layer fashion, that is both from the Network (NWK) and the Application Support sub-layer (APS). Specifically, the layer that originates a ZigBee frame is responsible for initially securing it. ZigBee frames can be both encrypted and authenticated. In case authentication is requested, an additional *Message Integrity Code* (*MIC*) is appended next to the frame payload. Note that only data frames and command frames can be secured, while ACK frames are always sent in the clear.

Octets: 1	4	0/8	0/1
Security control	Frame counter	Source address	Key sequence number

Table 3.1. ZigBee Auxiliary Security Header.

Protected frames include an additional *Auxiliary Security Header* (*ASH*) just before the payload. The structure of the ASH is shown in Table 3.1. The *Security control* field indicates by means of which key an outgoing (incoming) ZigBee frame has to be (is) secured, the MIC size in bytes (i.e. 0, 4, 8, 16) and whether or not the payload is encrypted. The *Frame counter* field is used to verify frame freshness and prevent processing of duplicate frames (*anti-replay*). When present, the *Source address* field indicates the extended 64-bit address of the device responsible for securing the frame. Finally, the *Key sequence number* field indicates the key sequence number of the Network Key possibly used to secure the current frame.

In order to activate security, the *nwkSecurityLevel* parameter must be set to a value greater than zero. ZigBee security is based on the reuse of cryptographic keys by different layers, in order to reduce storage costs. More specifically, in certain cases APS frames can be secured by means of security material maintained by the NWK layer. The APS layer deals also with key management (i.e. key establishment and key transport) and provides device management services. A *Trust Center* node, which typically coincides with the ZigBee Coordinator, handles key management and other security services.

In order to protect communication within a ZigBee network, up to five different kinds of cryptographic keys are supposed to be used.

- The *Network Key* is used to protect NWK frames. It is commonly shared within the whole ZigBee network, and is typically generated by the Trust Center. It is supposed to be periodically renewed.
- A *Data Link Key* can be shared between any two peers to protect APS frames.
- A *Key-Transport Key* is derived from a Data Link Key. It is used to protect messages carrying a Network Key, in case of Network Key distribution or renewal.
- The *Master Key* is a long term secret used during the Symmetric Key Key Establishment (SKKE) protocol. However, it is present only in High Security Mode.
- A *Key-Load Key* is derived from a Data Link Key. It is used to protect messages carrying a Master Key. As Master Keys, Key-Load Keys are present only in High Security Mode.

3.3.1 ZigBee NWK frame protection

If the *nwkSecureAllFrames* parameter is set to TRUE, security is applied to every incoming and outgoing NWK data frame. NWK frames are protected by means of the current Network Key. In particular, for NWK data frames, *securityEnabled* must be

specified as TRUE before starting a transmission. Figure 3.2 shows an example of protected ZigBee NWK data frame.

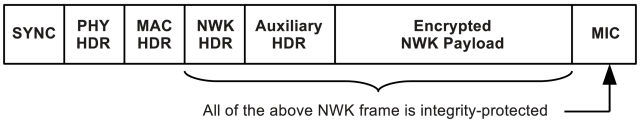


Figure 3.2. A protected ZigBee NWK frame.

3.3.2 ZigBee APS frame protection

ZigBee frames generated at the APL layer are protected according to their specific type. Figure 3.3 shows an example of protected ZigBee APS data frame.

- *APS data frames* are protected using either the Network Key or the Link Key shared with the recipient node. More specifically, the actual key to be used depends on the specific application profile which the particular data frame refers to.
- *Transport Key commands* are used to carry cryptographic keys. The key used to protect them must be selected according to the key to be transported and protected. For instance, Key-Transport Keys are used to protect Network Keys.
- *Other APS commands* are supposed to be protected by means of the Data Link Key shared with the recipient node. In case such a key could not be retrieved, the Network Key can be used. In the latter case, if the NWK layer is already applying security (i.e. the *nwkSecureAllFrames* parameter is set to TRUE), then the APS layer must not have to.

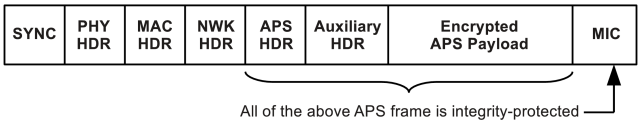


Figure 3.3. A protected ZigBee APS frame.

3.3.3 ZigBee security modes

As stated before, ZigBee provides two distinct security modes. The *Standard Security Mode* is designed for lower-security residential applications (e.g. Home Automation scenarios). The Trust Center device is supposed to maintain a Network Key and control network admittance policies. Also, the Network Key is either pre-installed on devices, or obtained unsecurely before joining the network. Potential Data Link Keys must be pre-installed, and almost no key management functionalities are provided.

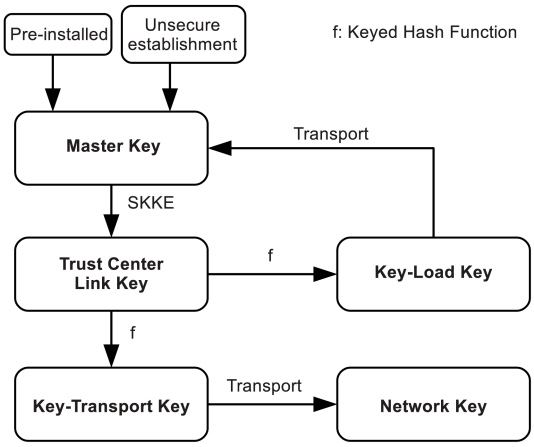


Figure 3.4. Key relationships in High Security Mode.

On the other hand, the *High Security Mode* is designed for high security commercial applications. The Trust Center is supposed to maintain a list of devices, Master Keys, Data Link Keys and Network Keys. In addition, the High Security Mode mandates devices authentication and Link Keys establishment with the Trust Center, by means of the *Symmetric Key Key Establishment* (*SKKE*) protocol, based on Elliptic Curve Cryptography [28]. According to the SKKE, two peers agree on a Master Key and exchange ephemeral quantities. Then they generate a common Data Link Key by means of a Key Derivation Function, and confirm with each other that the new key has been correctly established. Figure 3.4 summarizes the relationships among cryptographic keys in High Security Mode.

3.4 Security in ZigBee Smart Energy Profile

The ZigBee Smart Energy Profile (SEP) considers security as a major issue and includes precise mechanisms to secure communication, as well as a proper *Key Establishment Cluster* [117]. Besides, it refers to the ZigBee Standard Security Mode, but provides several improvements and enhancements.

In order to become part of a ZigBee network, a new device has to pass through a process called *Commissioning*, which is known as the task of configuring devices and networks to achieve the needs of the specific installation [34]. The ZigBee Alliance has recognized the importance of Commissioning and, in particular, the importance of specifications for Commissioning in a multi-vendor environment. According to SEP, devices can form their own network or join an existing network.

The Commissioning process is critical from a security viewpoint, but it has to be simple from a user perspective, and capable to provide some sort of feedback. Besides, it is assumed to be accomplished by a trusted person. A Commissioning Cluster

is currently under development [114]. However, from a very high level of abstraction, Commissioning consists of the following steps:

1. The network must be informed of the device that is to be joined. This operation is done through out-of-band means, which could include a web login, a phone call to a service center, or an interaction through an hand-held appliance. By means of this operation, the network is made aware of the device identifier (ID) and security information appropriate for the device.
2. The network is put into *permit joining ON* state.
3. The installer/homeowner is prompted to press a button or complete a menu sequence that tells the device to attempt to join the network.
4. The device attempts to join the network. In doing that, the device is authenticated using the appropriate security mechanisms and new keying material is distributed using the Key Establishment Cluster. In the following, we refer to this step as *Join procedure*.
5. An indicator is provided for the installer/homeowner indicating the device has joined the network and authenticated properly or provides information about improper authentication.
6. The device can now operate normally on the network.

Step 1 assures that the device is under the physical control of the installer user, and has been explicitly authorized by him to join the ZigBee network. The Join procedure is the most complex step and it is described in detail later in this section. Therefore, the Commissioning process seems to provide a pretty good robustness regarding device access to the network, thanks to the very first step and to the Key Establishment Cluster features.

3.4.1 Cryptographic keys

SEP assumes three kinds of keys: the Link Key, the Network Key, and the Key-Transport Key. A *Link Key* is an end-to-end key that a device may share with another device. However, a device must share a Link Key with the Trust Center (TC). This key is called *Trust Center Link Key* (*TCLK*). The end device and the Trust Center establish the TCLK during the Join procedure (step 4 of the Commissioning process). The TCLK is used to protect application level messages and stack commands. SEP allows the Trust Center to refresh the TCLK established with an end device, but suggests that it should be an infrequent operation.

A *Network key* is shared by all devices and is used to protect management and control communication. Also application level data and commands can be protected by means of the Network Key, in case either a Link Key can not be retrieved or the network layer is explicitly requested to secure outgoing frames.

Finally, every device shares a *Key-Transport Key* with the Trust Center. This key is derived from the TCLK, and its main use consists in securing the Network Key refresh process within the ZigBee network.

According to SEP, the Trust Center has to periodically refresh the Network Key. Such a rekeying is protected by means of the Key-Transport Keys. It follows that this rekeying is performed in a point-to-point way and thus its complexity in terms of rekeying messages amounts to $\mathcal{O}(n)$, where n is the number of devices in the network. In order to prevent potential de-synchronization problems, the Trust Center can order to start using the new Network Key by sending a proper SWITCH_KEY command to all devices after the new Network Key has been distributed all over the network.

3.4.2 Device authentication and key establishment

Now we focus on the security mechanisms which take place during the secure Join procedure. Basically, as specified at step 4 of the Commissioning process, an end device has to authenticate itself and exchange security information items with the Trust Center after it has joined the network. Specifically, a device has to obtain the current Network Key from the Trust Center, and establish a new end-to-end Trust Center Link Key with it. The key establishment process consists of the following steps:

1. *Establishing the TCLK*. The Key Establishment Cluster specifies that each device i has a pre-installed Trust Center Link Key LK_i, typically obtained from the device Installation Code, or similar. LK_i is provided to the local Trust Center through out-of-band means. This operation could take place at step 1 of the Commissioning process, while informing the Trust Center about the device that is to be joined.
2. *Establishing the Key-Transport Key*. The device i and the Trust Center can now obtain the Key-Transport Key TK_i deriving it from LK_i.
3. *Distributing the Network Key*. As soon as the device i has joined the network, the Trust Center sends it the Network Key NK encrypted by means of TK_i.
4. *Establishing a new Link Key*. As soon as the Join procedure has been completed, the Trust Center must update the Trust Center Link Key LK_i of the joining device i as described below.

According to the Key Establishment Cluster, during the Join procedure the key establishment process should follow the Certificate-Based Key Establishment (CBKE) method, since it provides the most comprehensive form of key exchange among two nodes in the network. Every device holds a certificate issued by a trusted Certification Authority (CA). Through such a certificate, it is possible to retrieve the device public key and other useful security information. The main reason that led to adopt the CBKE method is the need to safely identify a device, before it can start data communication.

The key establishment process between an initiator and a responder consists essentially in the following four steps:

1. Exchange Static and Ephemeral Data.
2. Generate Key Bitstream.
3. Derive Message Authentication Code (MAC) key and Key Data.
4. Confirm Key using MAC.

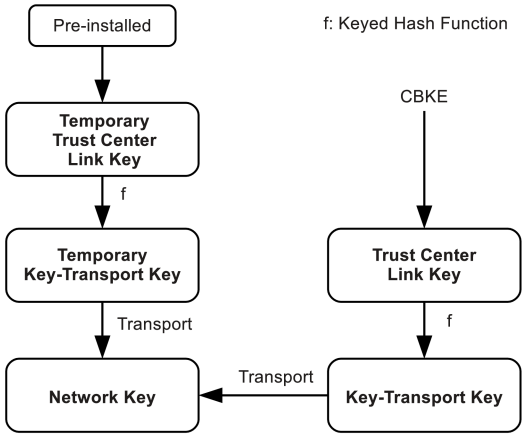


Figure 3.5. Key relationships in ZigBee Smart Energy Profile.

Regarding the second and third step, the key establishment procedure refers to the Elliptic Curve MQV key agreement scheme and a Key Derivation Function respectively, both described in [28]. At the end of this process, the Trust Center and the end device i share a new Link Key LK_i that is going to be used to protect data communication between them. Observe that the new Key-Transport Key TK_i is obtained from the new Link Key LK_i just established. Figure 3.5 summarizes the relationships among cryptographic keys in ZigBee Smart Energy Profile.

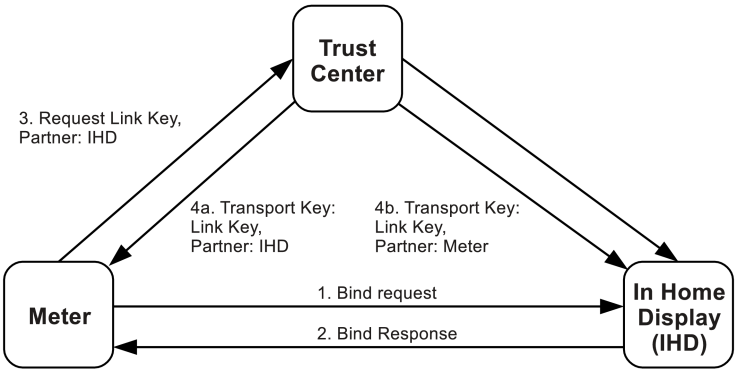


Figure 3.6. Pairwise Data Link Key establishment.

Once a device i has joined and been authenticated via key establishment and obtained an authorized Link Key LK_i with the Trust Center, it may need to communicate with another device j on the network, using application layer encryption. Rather than using key establishment between them, it would be advantageous to leverage the Trust Center to broker trust with other devices on the network. In fact, if two devices i and j have both obtained their Link Key with the Trust Center via key establishment,

then they both trust the Trust Center. So both devices use the Trust Center to request a Link Key with each other. The Trust Center responds to each node individually, sending a randomly generated Link Key LK_{ij}, protecting it by means of the respective Link Key LK_i and LK_j. Figure 3.6 summarizes the pairwise Data Link Key establishment.

3.4.3 Leaving the network

A device that has temporarily lost its connection to the network can perform a rejoin by means of a NWK *Rejoin procedure*. First, the device must attempt a *secured* rejoin, using the current Network Key. In case of failure (i.e. the Network Key has just been refreshed), it must attempt an unsecured rejoin, which will be successful only if the Trust Center has a Link Key with the device that was established using the Key Establishment Cluster. In case even the *unsecured* rejoin fails, the device has no other options but to repeat the standard Join procedure from the start. Finally, if a device i leaves the network, the Trust Center must remove the Trust Center Link Key LK_i assigned to that device.

3.5 Security concerns and possible solutions

In this section, we highlight security concerns we have found in the Smart Energy Profile (SEP), and propose possible solutions.

3.5.1 On supporting forward security

In general, a device leaves the network when it has accomplished its mission and thus it is dismissed or when it is momentarily sent to maintenance. Furthermore, a device may be forced to leave the network if it is compromised or suspected to be so. In any case, a device that has left the network must not be able to access any further communication in the network (forward security). Otherwise, if it ends up into an adversary's hands, she could abuse of the keying material still stored in the device. For this reason, the forward security requirement is typically achieved by a proper key revocation and redistribution (rekeying) policy [92]. In this section, we argue that SEP fails to specify a proper rekeying policy, so raising security and efficiency concerns.

As stated in Section 3.4.3, when a device i leaves, or is forced to leave, the network, the Trust Center must revoke the Trust Center Link Key LK_i assigned to that device. In order to do that, the Trust Center can simply delete that key. By doing so, the device will be unable to establish any further connection with the Trust Center, since it is not associated to any valid Link Key anymore.

However, SEP says nothing about the Network and Link Key management upon a device leaving. More precisely, a device that has left the network still retains the Network Key NK and all Link Keys LKs it established with peer devices. If these keys are not properly revoked and redistributed, the device remains able to overhear

and/or actively take part in all communication protected by means of these keys. In the case of the Network Key, this threat is particularly serious because the device would be able to access all communication protected by the NWK layer security, namely network and application layer commands and even application data messages, when allowed by the specific application profile. Thus, if the device is compromised, the adversary controlling it could exploit the Network Key to spoof and inject bogus routing information—e.g., false routes, bogus information about network status and link conditions—and perform highly disruptive routing attacks such as the sinkhole and selective forwarding attack [24].

As to the Network Key, the ZigBee Specification dictates that the Trust Center must refresh such a key periodically, but it neither clarifies how to determine the refresh period nor, more importantly, specifies any event that asynchronously triggers the Network Key refresh. This implies that an implementation that does not refresh the Network Key upon a node's leaving, and thus becomes exposed to the aforementioned threats, would be still compliant with ZigBee Specification.

A possible solution consists in revoking the Network Key every time a device leaves and redistributing a new one to all remaining nodes. Rekeying also assures that a device which has left is not able to perform a secured rejoin, being forced to employ the Key Establishment Cluster procedure to rejoin the network.

In fact, ZigBee provides two ways to refresh the Network Key: broadcast-based refresh and unicast-based refresh. In the *broadcast-based* refreshing, the new Network Key NK^+ is protected by means of the current Network Key NK. This is certainly a suitable solution for protecting the Network Key periodic refreshing against an external adversary. However, it is not acceptable for refreshing the Network Key upon a device's leaving of the network. Actually, in this case the current Network Key NK is compromised and cannot be trusted anymore. In such a case, the *unicast-based* refreshing can be used instead. In the unicast-based refreshing, the new Network Key NK^+ is delivered to every device i in a one-to-one fashion, protecting it by means of the device's Key-Transport Key TK_i. Although this solution is secure, it clearly has scalability limitations, due to the amount of encryptions and rekeying messages that grows according to $\mathcal{O}(n)$, where n is the number of devices in the network [92].

As to the Link Key between two devices, neither the ZigBee Specification nor SEP specify how to deal with it when one of the two devices leaves the network. Thus, it would be reasonable and wise to invalidate also all Link Keys that a leaving node has established with every other peer device which is still a member of the ZigBee network. However, ZigBee provides no mechanisms to explicitly inform a device that another one has left the network.

It must be said that ZigBee provides a mechanism to notify that a device is about to leave the network. However, such a mechanism is designed just to assure that network activity, basically the routing process, can be kept alive after a device has left the network. Also, an application layer command is present, but it is meant to be used by Routers to inform just the Trust Center that another device has a status that needs

to be updated (i.e. it left the network). So, as a matter of fact, ZigBee does not provide an explicit way to inform nodes about a device which has left the network.

A possible solution to efficiently and securely managing rekeying could be at the application level, so avoiding the security features directly provided by ZigBee. However, as clarified within ZigBee Specification, every application level protocol message requires proper identifiers for the presently considered application profile, cluster and command, each one with its specific payload. Thus, the introduction of an application level protocol might involve the extension of existing clusters with new commands or, as an alternative, even the definition of a brand-new manufacturer-specific cluster.

That being said, we claim it is worth defining and introducing a new ZigBee *Key Revocation Cluster*, aimed at coping with devices removal and capable to provide an efficient and secure Link Key and Network Key revocation and redistribution procedure. This cluster considers rekeying in two steps. The first rekeying step deals with Network Key revocation and redistribution. There are many network rekeying protocols properly conceived for networks composed of low-power, low-rate devices [41][44][47][69][111]. The advantage of implementing network rekeying at the application level consists in letting the application/system developer choose the rekeying scheme most suitable to the specific scenario requirements and constraints. The second rekeying step deals with the Link Key revocation and leverages on the previous step. After a device i has left the network and the Network Key has been revoked and redistributed, the Trust Center notifies every remaining device of that event by means of a broadcast message authenticated by the new Network Key. Upon receiving the notification message, each device $j, i \neq j$, can verify if it is sharing a Link Key LK_{ij} with i, and, if this is the case, discard it.

3.5.2 On supporting backward security

In order to guarantee backward security as well, it would be wise to refresh the current Network Key NK each time a new device i is about to join the ZigBee network. If we exclude the presence of malicious nodes within the network, it is sufficient to broadcast a new Network Key NK^+ to all present devices, protecting it via the current Network Key NK. Then, every node will start using NK^+ as the current Network Key. Once this procedure has been completed, the new device i is allowed to securely join the network, and the Trust Center provides it with the Network Key NK^+, as described in Section 3.4.

3.5.3 Certificate management

As discussed in Section 3.4.2, the key establishment process follows the Certificate-Based Key Establishment (CBKE) scheme. This implies that every device holds a certificate issued by a Certification Authority (CA). In order to generate certificates and verify their validity, SEP refers to the ECQV Implicit Certificate Scheme [29].

CBKE provides every device with an implicit certificate and the public key of the CA releasing the certificate, also called the CA *root key*. Implicit certificates include neither the subject public key nor a traditional CA's signature. Thereby they are supposed to be smaller than conventional certificates, as well as more efficient to handle, since there is no signature to be verified. However, they make it possible to compute the certified public key, which is retrieved by means of the CA root key.

ZigBee Key Establishment Cluster claims that many different subjects can issue certificates, namely device manufacturers, device distributors, and even end-customers. However, it seems to underestimate the management issue that ensues from the limited storage resources on ZigBee end devices. Actually, in order that any Coordinator and any end device, possibly coming from different manufacturers or different distributors, can inter-operate, it is necessary that they are able to authenticate each other. This requires that the one is able to verify the other's certificate. It follows that each device should store the root key of every possible certification authority releasing implicit certificates for devices. While we can reasonably assume that a Coordinator has no storage limitations, and thus can keep a large set of root keys, the same does not hold for end devices that have scarce storage resources. Of course, at the other extreme of the spectrum of solutions, we can assume the existence of a single certification authority, or, at least, a very limited number, so as to manage a meager number of root keys. However, practice proves that this approach is pretty unrealistic and might lead to a monopoly regime.

In order to provide a practical solution to this problem, we introduce another level of certification. We assume that the network administrator, i.e. the installer or the homeowner, runs a *Home Certification Authority* (CA_H) which stores the root keys of certification authorities releasing implicit certificates for devices. We call this set of keys the *Root Keys Database*. The task of the Home Certification Authority consists in verifying the certificate pre-installed in a device and, if the verification succeeds, issuing a new certificate for that device. We call this process *home-certification*.

More in detail, let D be a device, K_D its public key and $\langle D \rangle_{CA}$ a certificate released to D by a given certification authority CA (see Figure 3.7). The certificate is pre-installed in the device. The Home Certification Authority home-certificates the device according to the following steps.

1. The Home Certification Authority CA_H obtains the device's certificate $\langle D \rangle_{CA}$.
2. CA_H retrieves the CA's root key from the Root Keys Database and verifies $\langle D \rangle_{CA}$ by means of that key (if the key is not present, CA_H obtains it from the Internet and updates the database).
3. CA_H issues a new home-certificate for D, namely $\langle D \rangle_{CA_H}$.
4. The new home-certificate $\langle D \rangle_{CA_H}$ is installed in the device D.

If the home-certification is applied to the Trust Center at the moment the network is started up, later any device that joins the network needs only to know and store K_{CA_H} in order to authenticate the Trust Center certificate $\langle TC \rangle_{CA_H}$. So doing, the

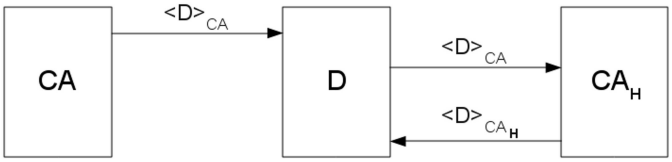


Figure 3.7. Home-certification.

storage demand for authentication is drastically reduced to just one key. The most reasonable choice is to provide a device with K_{CA_H} during the execution instance of the Commissioning process for that device, and before the Join procedure is carried out (see Section 3.4). Observe that devices can trust K_{CA_H} since it has been created by CA_H which, in turn, is managed by the network manager. So, trustworthiness of K_{CA_H} is clearly related to the network manager capability to manage CA_H.

If the home-certification is applied not only to the Trust Center but also to other devices, then also the Trust Center needs to store only the public key K_{CA_H} of the Home Certification Authority CA_H to authenticate every device. Thus, storage saving may be also at the Trust Center side. Finally, we observe that even the Trust Center can act as the Home Certification Authority CA_H. However, in this case it should have to store the Root Keys Database, so losing every benefit in terms of storage saving deriving from the home-certification of devices.

The proposal discussed above introduces some new challenges: the user would have in his own hands a core component of the security architecture. Therefore, she should be capable of managing her home CA and issuing new certificates. Thus, this approach could evidently turn out to be in contrast with the simplicity recommendations provided by ZigBee Specification and ZigBee application profiles about user duties within the network. On the other hand, it is evident the lack of clear specifications about how to organize the Public Key Infrastructure, in order to validate device certificates. Nevertheless, it is really important to focus on assuring simplicity to the final user while coping with that.

4

Key Management in secure group communication

Group communication has proven to be a suitable and efficient communication model for distributed systems in many different application scenarios, including content distribution, Wireless Sensor Networks (WSNs), teleconferencing, wargaming, and others. These application scenarios typically involve a large number of participants, which dynamically join and leave the application, so causing group membership to change frequently. Furthermore, they also require that group communication is protected from unauthorized accesses. This is achieved by restricting the access to group communication to group members only.

Intuitively, this is achieved by distributing a *group key* to all group members which use it to encrypt (decrypt) messages broadcast (received) within the group. Good cryptographic practices suggest to periodically refresh the key in order to prevent cryptanalysis. Furthermore, when a new user joins the group, she must not be able to access any group communication prior its joining (*backward security*). Besides, when a current member leaves the group, she must be prevented from accessing any further group communication (*forward security*). As a consequence, upon a new user's joining or a current one's leaving, the current group key has to be revoked and a new one has to be distributed. This process is known as *rekeying*, and makes it possible to fulfill the backward and forward security requirements [92].

Rekeying the group upon a new member's joining is actually trivial. In contrast, rekeying the group upon a current member's leaving is a far more complex problem. A naïve solution consists in separately rekeying every remaining member, by transmitting that member the new group key encrypted with the member's secret key. Although very simple, this solution requires $\mathcal{O}(n)$ rekeying messages, where n is the group size, and thus scales poorly. Actually, efficient rekeying requires to broadcast the new group key in its encrypted form into the group. However, the current group key cannot be used, because the leaving member is aware of it. A typical approach is to encrypt rekeying messages by means of administrative keys that are not known to the evicted member. Furthermore, the administrative keys known to the evicted member

should also be replaced. It follows that efficient rekeying is a crucial component in any secure group communication system [13][23][33][72][73][77][92][96].

A group member may leave a group for several reasons. For instance, because its mission is concluded, its subscription is expired, or it has failed or depleted its energy. Also, a group member may be forced to leave because it has been compromised or it is suspected so. A *collusion attack* occurs when evicted compromised members share their security material in order to regain access to the group key [92]. A typical collusion attack consists of an adversary capturing a set of users, incrementally aggregating the uncovered keying material of individual members to a level that allows revealing the group key and, consequently, all encrypted traffic in the network. No group rekeying scheme is exempt from collusion attacks, and different schemes display different levels of resilience [33][55][56][67][68][71]. After a successful collusion attack, group rekeying schemes generally require a *total group recovery*, i.e. every group member must be re-initialised in a one-to-one fashion. Of course, such a total group recovery has a negatively impact on the overall rekeying performance.

In this chapter, we describe *HISS* [47], a highly scalable rekeying scheme for large scale, dynamic groups. HISS is *centralized* and levers on *logical subgrouping*, a consolidated conceptual technique that has been already exploited in several existing group rekeying schemes [23][33][44][55][67][68][70][71]. HISS partitions group members into non overlapping logical subgroups that become the units of rekeying and recovery from collusion attacks. With respect to other rekeying schemes based on subgrouping, HISS features two novel contributions.

First, unlike other well-known centralized approaches [23][33][44][55], HISS rekeys the system with a number of messages that is small, constant, and independent of the group size, thus resulting highly scalable and efficient. Specifically, HISS only requires *two* broadcast messages to rekey the system, so displaying $\mathcal{O}(1)$ communication complexity. One broadcast message rekeys the subgroup containing the leaving member, and another one rekeys all the remaining subgroups.

Second, differently from [67][68][70][71], HISS considers collusion attacks as a first-class problem, and provides an integrated *recovery protocol* that re-establishes security as soon as a collusion attack has been detected. Such a protocol gracefully decreases in efficiency with the collusion attack severity, and does not require a total group recovery of the system. In fact, only compromised subgroups are totally recovered, whereas uncompromised ones are efficiently rekeyed by a single broadcast message each. The rational basis for this is that while rekeying accounts for the normal functioning of the system (joining or a leaving), recovery has to be considered an exceptional event. Therefore, according to the well-known Lampson's recommendations for computer systems design [19], while rekeying must be very efficient, it is sufficient that recovery is able to make progress provided it remains practically sustainable. Also, we show that a proper allocation of users to subgroups may *practically* increase the resilience of HISS to collusion attacks or, even, prevent them altogether.

4.1 System architecture

We consider a group G of users. A user becomes member of the group by explicitly *joining* it. As a member of the group, a user may broadcast messages to other members. Later, a member may voluntarily *leave* the group or be *forced* to leave if compromised or suspected to be so. In order to guarantee security of group communication, it is generally required that when a user joins the group it is not able to access group communication prior its joining (*backward security*), and that when a user leaves the group it is prevented from accessing any further group communication (*forward security*). In order to achieve this, group members share a secret cryptographic key, the *group key*, which they use to encrypt and decrypt messages transmitted and received, respectively, within the group. We denote by K_G the group key of the group G. When a new user joins or a current member leaves the group, the current group key is revoked and a new one is distributed. We call this operation *rekeying*.

The group is managed by a *Group Controller* (GC), which is composed of three different components: a *Group Membership Service* (GMS), a *Key Management System* (KMS), and an *Intrusion Detection System* (IDS). The GMS maintains group membership by keeping track of users that join and leave the group. A user wishing to join the group invokes the join operation. Later, a user may leave the group by invoking the leave operation. As they are exposed to attackers, the IDS monitors network activities to detect compromised users. As there exist no sure and efficient way to readily detect a single user capture [68][94], the IDS may return multiple compromised users at once. Upon detecting them, the IDS invokes the leave operation specifying the set G_c of compromised users, in order to have them evicted from the group. The IDS component is beyond the scope of this dissertation, and readers may refer to [18][38][61][83][93][109] for more details.

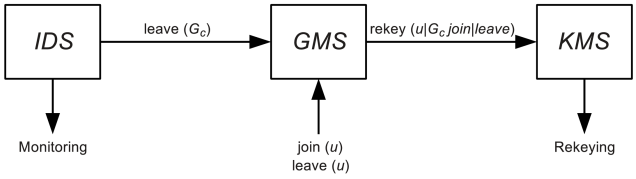


Figure 4.1. The Group Controller GC.

Whenever a user joins, leaves or is evicted, the group key has to be renewed in order to guarantee the backward and forward security requirements. The KMS is responsible for the rekeying task. When managing a change in the group membership, the GMS activates a rekeying. In particular, it invokes the rekey(u, join) operation when a new user u joins the group, or the rekey(G_c, leave) operation when a set G_c of users have to leave the group. Figure 4.1 shows the architecture described above.

In a centralized approach, GC is implemented by a resourceful computing node that is generally more powerful than users. GC may be a server with plentiful of computing, storage, communication, and power resources. Furthermore, we assume that GC is trustworthy and thus cannot be compromised by attackers. Although server security is still a research issue, the literature provides a number of established techniques and methodologies to keep servers secure. For instance, good starting readings are [36][88]. In the following, we detail the *Key Manager* (KM) that implements the Key Management System in the centralized approach.

4.2 The rekeying scheme

In this section, we describe HISS. Specifically, in Section 4.2.1, we provide an informal description of the basic rekeying scheme with particular reference to key revocation and distribution upon a user's leaving. Then, in Section 4.2.2, we introduce the problem of collusion attack, and give an informal description of how HISS solves it. Finally, in Section 4.2.3, we present the rekeying protocols that manage a user's leaving and joining, as well as the recovery from multiple colluding user captures.

4.2.1 The basic scheme

We assume that the group G is *partitioned* into a set $\mathcal{S}$ of nonempty subgroups, such that every member of G is exactly in one of these subgroups. More formally, let $\mathcal{S} \subseteq 2^G$ be a *partition* of G. Then, $\forall S, S' \in \mathcal{S},\ S \cap S' = \emptyset$, and $\bigcup_{S \in \mathcal{S}} S = G$. We call *cognate* two users belonging to the same subgroup. Furthermore, we consider the function *SubgroupOf* : $G \rightarrow \mathcal{S}$ that returns the subgroup of a given user, i.e. $\forall u \in G, \forall S \in \mathcal{S}, S =$ *SubgroupOf*(u) $\text{iff} u \in S$. Subgroups are relevant only for key management and have no application-level meaning. Finally, we consider the function *SubgroupSetOf* : $2^G \rightarrow \mathcal{S}$ that given $G' \subseteq G$ returns a set of subgroups $\mathcal{S}'$ such that $\forall u \in G' \Rightarrow$ *SubgroupOf*$(u) \in \mathcal{S}'$ and $\forall S \in \mathcal{S}', S \cap G \neq \emptyset$.

We assume that every user and every subgroup are associated with random quantities called *tokens*. We call *user tokens* and *subgroup tokens* the tokens associated with users and subgroups, respectively. We denote by t_u the token associated to user u, and by t_S the subgroup token associated to subgroup S. We also assume that every user *a priori* shares a *user key* with the Key Manager KM. We denote by K_u the user key of user u. Finally, we assume that every subgroup is associated to a *subgroup key*. We denote by K_S the subgroup key of subgroup $S \in \mathcal{S}$. A subgroup key is shared between the Key Manager KM and every user in the subgroup. KM and users keep tokens and keys secret.

The Key Manager maintains subgroups, tokens and keys for all users and subgroups in the system. In particular, the Key Manager records: i) all user tokens in the User Token Set (UTS); ii) all subgroup tokens in the *Subgroup Token Set* (STS);

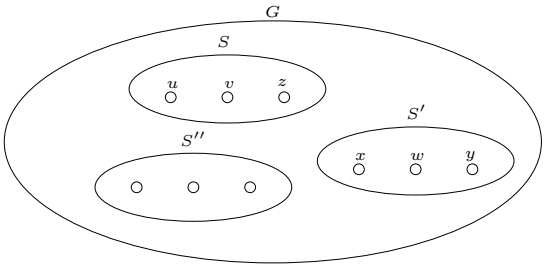
iii) all user keys in the *User Key Set* (UKS); and, finally, iv) all subgroup keys in the *Subgroup Key Set* (SKS).

Let us consider a user u belonging to a subgroup S, i.e. $S = \textit{SubgroupOf}(u)$. The user maintains its user key K_u and the subgroup key K_S associated with its subgroup S. Furthermore, the user maintains the User Token Set UTS_u, namely the set of user tokens associated to its cognate users. More formally,

$$UTS_u = \{t_v | v \in S \wedge v \neq u\}. \tag{4.1}$$

Finally, the user u maintains the Subgroup Token Set STS_S, i.e. the set of subgroup tokens of all subgroups belonging to the absolute complement of S in $\mathcal{S}$. More formally,

$$STS_S = STS_{\textit{SubgroupOf}(u)} = \{t_R | R \in \mathcal{S} \wedge R \neq S\}. \tag{4.2}$$



(A)

Subgroup	User	User Token Set	Subgroup Token Set	User Key	Subgroup Key
S	u	$\{t_v, t_z\}$	$\{t_{S'}, t_{S''}\}$	K_u	K_S
	v	$\{t_u, t_z\}$	$\{t_{S'}, t_{S''}\}$	K_v	K_S
	z	$\{t_u, t_v\}$	$\{t_{S'}, t_{S''}\}$	K_z	K_S
S'	x	$\{t_w, t_y\}$	$\{t_S, t_{S''}\}$	K_x	$K_{S'}$
	w	$\{t_x, t_y\}$	$\{t_S, t_{S''}\}$	K_w	$K_{S'}$
	y	$\{t_x, t_w\}$	$\{t_S, t_{S''}\}$	K_y	$K_{S'}$

(B)

Figure 4.2. A) a group G partitioned in three subgroups S, S', and S'' and B) the keying material held by members of S and S'.

In order to fix ideas, without any ambition of generality, consider the example in Figure 4.2. Figure 4.2-A shows a group G partitioned in three subgroups S, S', and S'', while Figure 4.2-B shows data structures maintained by members of S and S'.

Intuitively, the Key Manager uses tokens to rekey the system in a scalable way as follows. With reference to Figure 4.2-A, let us suppose that user $u \in S$ leaves the group G and, therefore, the current group key K_G has to be revoked and a new one K_G^+ redistributed to all members of G but u. In order to distribute the new key to all

members of S but u, the Key Manager uses t_u. By construction, all users in S but u know t_u (see Figure 4.2-B). Therefore, the Key Manager can use t_u to generate a key encryption key, use it to encrypt K_G^+, and then broadcast the resulting ciphertext into S. All members of S but u can use t_u to derive the key encryption key, decrypt the received ciphertext, and, finally, obtain K_G^+.

In order to distribute the new key to all the other subgroups, the Key Manager uses t_S. By construction, this token is known to all members of any subgroup S' different from S. Thus, user u does not know t_S (see Figure 4.2-B). Therefore, the Key Manager can use t_S to generate another key encryption key, use it to encrypt K_G^+, and then broadcast the resulting ciphertext into G. Any user $w \in S', S' \neq S$, can derive the key encryption key, decrypt the received ciphertext, and, finally, obtain K_G^+.

4.2.2 Dealing with collusion attacks

Collusion attack is a typical problem in group rekeying [92]. We have a collusion attack when evicted members share their individual piece of information to regain access to the group key. Colluding users may be evicted malicious group members working together or compromised members under the control of the same adversary. In the worst case, a collusion attack requires a *total group recovery*, i.e. every single user of the group has to be re-initialized separately in a one-to-one fashion, thus limiting efficiency and scalability of the rekeying scheme.

If two or more users are captured within a subgroup, two cases need to be considered: 1) non colluding user captures (e.g. attacks carried out by different adversaries); and 2) colluding user captures. In the former case, it is possible to exploit the basic scheme in order to evict non colluding users (see Section 4.2.1). In the latter case, colluding attackers may compromise the whole subgroup. Actually, by joining the user token sets of at least two users it is possible to obtain the whole set of user tokens associated with the subgroup. With reference to Figure 4.2-A, if users u and v collude, then all user tokens associated to S, namely $UTS_u \cup UTS_v = \{t_u, t_v, t_z\}$, are compromised. Thus, user tokens cannot be used to rekey the compromised subgroup.

Notice that, in this case, subgroup tokens can still be used to rekey the other non compromised subgroups. However, in the most general case, multiple users belonging to two or more subgroups collude. Let us suppose that u, v, w, and x collude. Similarly to before, all user tokens associated with subgroups S and S' are compromised. However, with respect to the previous case, the adversary may obtain all the subgroup tokens by joining the Subgroup Token Sets of any two users belonging to different subgroups. With reference to Figure 4.2-A, if users $u \in S$ and $w \in S'$ collude, then all the subgroup tokens, namely $STS_S \cup STS_{S'} = \{t_S, t_{S'}, t_{S''}\}$, are compromised. Thus, subgroup tokens cannot be used to rekey non compromised subgroups.

The basic scheme described in Section 4.2.1 is efficient as it requires just two broadcast messages to rekey the group upon a member's leaving, and works well provided that captured users are non colluding. However, in practice collusion attacks

can occur, and the group key management system solution has to take this threat into account. Researchers have pointed out that there exist no sure and efficient way to readily detect a single user capture [68][94]. Therefore, for a group key management solution to be truly effective in a hostile environment, it must be able to recover from multiple user captures.

Let us suppose that r subgroups are compromised due to a collusion attack involving multiple users. Let $\mathcal{C} \subseteq \mathcal{S}$ be the set of these subgroups. Let $\mathcal{U}$ be the absolute complement of $\mathcal{C}$ in $\mathcal{S}$, i.e. $\mathcal{U} = \mathcal{S} \setminus \mathcal{C}$. Subgroups in $\mathcal{U}$ are not compromised. Then, every compromised subgroup in $\mathcal{C}$ needs to be totally recovered by unicasting the new keying material to every non captured user using its respective user key. Furthermore, every non compromised subgroup in $\mathcal{U}$ is recovered by broadcasting the new keying material to the subgroup using its subgroup key.

As it turns out, a collusion attack affects subgroups separately, and does not compromise the entire group. More specifically, a total group recovery is not necessary upon a collusion attack. In contrast, total recovery is necessary only for the subgroups involved in the attack. The other subgroups that are not involved in the attack can be efficiently rekeyed by broadcasting the new key material. This form of *partial recovery* provides a form of *graceful degradation* of performance, in terms of number of recovery messages and cryptographic operations.

4.2.3 The protocols

In this section, we present the leave, join, and recovery protocols in a more detailed way. Each protocol is a master-slave protocol, where the Key Manager is the master, and users are the slaves. Every user is structured as a collection of *message handlers*. Each handler is denoted by □ HANDLER *handler name*. The execution of a handler is triggered by the reception of the corresponding message, and runs uninterrupted until its completion.

We also use the following notation. By $h()$ we denote a one-way hash function [10]. A one-way hash function $h()$ is a function that has the following properties: given an input x, it is easy to compute the image $y, y = h(x)$, whereas given y it is computationally unfeasible to compute the preimage x so that $y = h(x)$. Furthermore, given an input x and its image $y = h(x)$ it is computationally unfeasible to find a second preimage z such that $y = h(z)$. Examples of one-way hash functions are SHA-1, SHA-2, and SHA-256 [79]. Furthermore, by $kdf()$ we denote a key derivation function, that is a pseudo-random function that derives one or more cryptographic keys from a secret value [15]. Keyed cryptographic hash functions are popular examples of pseudo-random functions used for key derivation. Finally, by $\{x\}_k$ we denote the encryption of quantity x by means of key k. Concerning this, we assume that: i) the cipher is computationally secure; ii) the cryptographic keys are generated by means of a secure random generator, unless otherwise specified; and iii) key length is adequate to discourage an exhaustive key search [10]. Finally, by $u \rightarrow v : m$ we denote

process u sending a unicast message m to process v, whereas by $u \twoheadrightarrow G : m$ we denote process u sending a broadcast message m to the group G.

System initialization

Upon *system initialization*, the Key Manager performs the following actions: i) randomly generate user tokens and subgroup tokens, as well as user keys and subgroup keys for all users and subgroups in the system; ii) store tokens and keys into the Key Manager's User Token Sets, Subgroup Token Sets, User Key Set, and Subgroup Key Set, as appropriate (see Section 4.2.1); and iii) initialize every user with the corresponding user and subgroup key as well as the user and subgroup tokens, according to the subgroup the user belongs to (see Section 4.2.1). User initialization (step iii) is performed through a pre-existing secure, confidential, and authentic channel.

Forward security

In order to assure forward security, the Key Manager KM relies on two different protocols, namely the *leave protocol* and the *recovery protocol*. The former protocol is used when a single user has to be evicted from the group G. This may occur when the user has completed its mission and thus leaves the group. Alternatively, this may happen when the Intrusion Detection System has detected a single user capture and thus the compromised user has to be evicted from the group. The recovery protocol is used in case the Intrusion Detection System has detected multiple possibly colluding user captures and thus multiple users have to be evicted at the same time.

More specifically, upon receiving the rekey(G_c, leave) call from the Group Membership Service GMS, the Key Manager KM determines whether to trigger the *leave protocol* or the *recovery protocol*, according to the following steps.

1. If G_c contains a single user u, then KM triggers the *leave protocol* specifying u as argument. Otherwise,

2. if G_c contains t users, $t > 1$, then
   a) if the t users belong to different subgroups, i.e. there are no couples of cognate users, then for each user $u \in G_c$, KM triggers an instance of the *leave protocol* specifying u. Otherwise,
   b) if a number of users in G_c are cognate, i.e. one or more subgroups have been compromised, then KM triggers the *recovery protocol* specifying the set G_c as argument.

It is worthwhile to notice that multiple user captures require the execution of the recovery protocol if and only if the compromised users are colluding. Actually, the presence of multiple user captures does not imply in itself that they are colluding too. If compromised users are not colluding, they can be efficiently evicted from the system by means of the leave protocol. In general, determining whether two or more

compromised users are also colluding may be a difficult task that strictly depends on the specific application scenario. A conservative application-independent policy, trading performance for security, could consist in invoking the recovery protocol whenever the Intrusion Detection System detects multiple user captures.

In the following, we abstract away from both the application scenario and the corresponding intrusion detection technique and recovery policy, and present the leave and recovery protocols in the two respective relevant situations, namely upon a user's leaving, or being evicted, and upon evicting a set of colluding captured users. These two protocols constitute the basic mechanisms for any intrusion management policy.

The leave protocol

Let us suppose that a user u, belonging to the subgroup S, leaves, or is forced to leave, the group G. The Key Manager KM revokes the current group key K_G and distributes a new one K_G^+ according to the *leave protocol*.

KEY MANAGER KM

1. KM generates i) a new group key K_G^+; ii) a new subgroup key K_S^+; iii) a key encryption key $KEK_u, KEK_u \leftarrow kdf(t_u)$, to rekey the subgroup S; and iv) a key encryption key $KEK_S, KEK_S \leftarrow kdf(t_S)$, to rekey all the other subgroups. Then,

2. KM broadcasts the following rekeying messages.

$$\begin{array}{l} \mathrm{M1} : KM \twoheadrightarrow S : u, \{K_G^+, K_S^+\}_{KEK_u} \\ \mathrm{M2} : KM \twoheadrightarrow G \setminus S : S, \{K_G^+\}_{KEK_S} \end{array}$$

   Finally,

3. KM installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$; installs K_S^+ as the current subgroup key of the subgroup S, $K_S \leftarrow K_S^+$; removes t_u from UTS, $UTS \leftarrow UTS \setminus \{t_u\}$, and updates the user tokens related to the remaining users in S, $\forall t_v \in UTS, v \in S, t_v \leftarrow h(t_v \| K_G^+)$; and, finally, updates all the subgroup tokens, $\forall t \in STS, t \leftarrow h(t \| K_G^+)$.

USER v

□ HANDLER LH1. Upon receiving message M1, user $v, v \in S, v \neq u$, performs the following actions.

1. User v computes the key encryption key $KEK_u, KEK_u \leftarrow kdf(t_u)$, and uses it to retrieve the new group key K_G^+ and the new subgroup key K_S^+. Then,

2. user v installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$; installs K_S^+ as the current subgroup key of the subgroup S, $K_S \leftarrow K_S^+$; removes t_u from UTS_v, $UTS_v \leftarrow UTS_v \setminus \{t_u\}$, and updates the remaining user tokens, $\forall t \in UTS_v, t \leftarrow h(t \| K_G^+)$; and, finally, updates its Subgroup Token Set STS_S, $\forall t \in STS_S, t \leftarrow h(t \| K_G^+)$.

□ HANDLER LH2. Upon receiving message M2, user $v \in S', S' \in \mathcal{S}, S' \neq S$, performs the following actions.

1. User v computes the key encryption key $KEK_S, KEK_S \leftarrow kdf(t_S)$, and uses it to retrieve the new group key K_G^+. Then,

2. user v installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$, and updates its Subgroup Token Set, $\forall t \in STS_{S'}, t \leftarrow h(t \| K_G^+)$.

It is worth noting that, in any execution of the leave protocol, a user v receives either message M1 or message M2. Since both the Key Manager and any user's handler have a fixed number of steps, the protocol time complexity is constant.

The recovery protocol

Let G_c be the set of compromised users. KM revokes the current group key K_G and distributes a new one K_G^+ according to the *recovery protocol*.

KEY MANAGER KM

1. Initially, KM builds $\mathcal{C} = \textit{SubgroupSetOf}(G_c)$ and $\mathcal{U} = \mathcal{S} \setminus \mathcal{C}$. Then, $\forall u \in G_c$, KM removes u from G and from $\textit{SubgroupOf}(u)$. Then, KM randomly generates i) a new group key K_G^+; ii) a new subgroup key K_S^+ for each compromised subgroup $S \in \mathcal{C}$; iii) a new user token t_u for each non compromised user u belonging to a compromised subgroup S, i.e. $u \notin G_c, u \in S, S \in \mathcal{C}$; and, finally, iv) a new subgroup token t_S for every subgroup $S \in \mathcal{S}$. Then,

2. $\forall S \in \mathcal{C}, \forall u \in S$, KM sends u a unicast rekeying message M_u

$$\mathrm{M_u} : KM \rightarrow u : \{K_G^+, K_S^+, UTS_u^+, STS_S^+\}_{K_u}$$

carrying the new group key K_G^+, the new subgroup key K_S^+, the new User Token Set UTS_u^+ (see equation 4.1) and the new Subgroup Token Set STS_S^+ (see equation 4.2) containing, respectively, the new user tokens and the new subgroup tokens built at step 1.

3. Then, $\forall S \in \mathcal{U}$, KM broadcasts a rekeying message M_S

$$\mathrm{M_S} : KM \twoheadrightarrow S : \{K_G^+, STS_S^+\}_{K_S}$$

carrying the new group key K_G^+ and the new Subgroup Token Set STS_S^+ (see Equation 4.2) which contains the subgroup tokens built at step 1. Finally,

4. KM installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$; installs the new subgroup keys as the current subgroup keys of each colluding subgroup belonging to $\mathcal{C}$; and, finally, updates its own User Token Set UTS and Subgroup Token Set STS with the user tokens and subgroup tokens generated at step 1.

USER u

□ HANDLER RH1. Upon receiving the unicast rekeying message $\mathrm{M_u}$, non compromised user u belonging to a compromised subgroup $S \in \mathcal{C}$ performs the following actions.

1. User u uses the user key K_u shared with KM to decrypt message $\mathrm{M_u}$ and retrieve the new group key K_G^+, the new subgroup key K_S^+, the new User Token Set UTS_u^+ and the new Subgroup Token Set STS_S^+. Then,

2. user u installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$; installs K_S^+ as the current subgroup key of the subgroup S, $K_S \leftarrow K_S^+$; and, finally, updates its own User Token Set, $UTS_u \leftarrow UTS_u^+$, and Subgroup Token Set, $STS_S \leftarrow STS_S^+$.

□ HANDLER RH2. Upon receiving the rekeying message $\mathrm{M_S}$, user u belonging to a non compromised subgroup $S \in \mathcal{U}$ performs the following actions.

1. User u uses the subgroup key K_S to decrypt $\mathrm{M_S}$ and retrieve the new group key K_G^+, and the new Subgroup Token Set STS_S^+. Then,

2. user u installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$, and the new Subgroup Token Set STS_S^+, as the current Subgroup Token Set, i.e. $STS_S \leftarrow STS_S^+$.

It is worth noting that, in any execution of the recovery protocol, a user u receives either message $\mathrm{M_u}$ or message $\mathrm{M_S}$. Also, involved users execute their own rekeying operations independently from one another. Since both the Key Manager and any user's handler have a fixed number of steps, the protocol time complexity is constant.

Note that in the presence of $\mathcal{C} \subset \mathcal{S}$ compromised subgroups, then $\mathcal{U} \neq \emptyset$, and the whole group G is not entirely compromised. Therefore, in order to restore secure communications, it is not necessary to totally recover it by unicasting rekeying messages to every non compromised users remained in the group G. In fact, it is necessary to unicast rekeying messages only to non compromised users in compromised subgroups, i.e. subgroups in $\mathcal{C}$. As subgroup keys associated to non compromised subgroups, i.e. subgroups in $\mathcal{U}$, are not compromised, then we can use each one of them to efficiently rekey its respective subgroup by means of a single rekeying message.

Backward security

Before a user can become a new member of the group G, the Key Manager has to refresh the network security material in order to assure backward security. In fact, a new user must not be able to gain knowledge of group activities that took place before its join, that is, it must be prevented from having access to old secured messages.

Let us suppose that a user u wants to join the group G and become a member of the subgroup $S \in \mathcal{S}$. First, u invokes the join(u) operation provided by the Group Membership Manager GMS. Then, GMS invokes the rekey(u, join) operation provided by the Key Manager KM. By doing so, KM revokes the current group key K_G and distributes a new group key K_G^+ according to the *join protocol*.

KEY MANAGER KM

1. KM generates a new group key K_G^+, a new subgroup key K_S^+, and a new user token t_u. Then,

2. KM broadcasts the following rekeying messages.

$$\begin{aligned} &\mathrm{M1} : KM \twoheadrightarrow G : \{K_G^+\}_{K_G} \\ &\mathrm{M2} : KM \twoheadrightarrow S : \{K_S^+, t_u\}_{K_S} \end{aligned}$$

   Finally,

3. KM installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$; installs K_S^+ as the current subgroup key of the subgroup S, $K_S \leftarrow K_S^+$ and, finally, adds t_u to UTS, $UTS \leftarrow UTS \cup \{t_u\}$.

USER v

- HANDLER JH1. Upon receiving the rekeying message M1, user $v \in G$ performs the following actions.

  1. User v uses K_G to retrieve the new group key K_G^+. Then,

  2. user v installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$.

- HANDLER JH2. Upon receiving the rekeying message M2, user $v \in S$ performs the following actions.

  1. User v uses K_S to retrieve the new subgroup key K_S^+ and the new token t_u. Then,

2. user v installs K_S^+ as the current subgroup key of the subgroup S, $K_S \leftarrow K_S^+$, and adds t_u to UTS_v, $UTS_v \leftarrow UTS_v \cup \{t_u\}$.

Once the join protocol has been completed, user u joins the group G and the subgroup S, according to the following steps.

1. KM provides user u with the new group key K_G^+, the new subgroup key K_S^+, the $(m-1)$ user tokens associated to its subgroup cognates, and the $(p-1)$ subgroup tokens $st_{S'}$, $S' \neq S$, through a pre-existing secure channel. Then,

2. user u installs K_G^+ as the current group key, $K_G \leftarrow K_G^+$; installs K_S^+ as the current subgroup key of the subgroup S, $K_S \leftarrow K_S^+$; and, finally, builds its own User Token Set UTS_u (see equation 4.1) and Subgroup Token Set STS_S (see equation 4.2) with the user tokens and subgroup tokens received at step 1.

As stated at step 1, the keying material is provided to the new user u by the Key Manager through a pre-existing secure channel. As a consequence, authentication and confidentiality are both assured. Possible implementations include an a priori shared cryptographic key or out-of-band means.

4.3 Security analysis

In this section, we argue that the leave, recovery, and join protocols guarantee the forward and backward security requirements.

Theorem 1. *The leave protocol guarantees forward security.*

Proof. Given the assumptions on the cipher strength and key length, the proof consists in showing that all group users but $u \in S$ can derive the next group key K_G^+ from an execution instance of the leave protocol, and that user u cannot derive it nor any future group key.

Thanks to HANDLER LH1, all users in S but u retrieves the new group key K_G^+. Then, thanks to HANDLER LH2, every user $v \in S', \forall S' \in \mathcal{S}, S' \neq S$, retrieves K_G^+. Thus, u is evicted from the group G, and all other users hold the new group key K_G^+.

Also, tokens are updated by KM and users in S by means of the new group key K_G^+. As it does not hold such a key, u cannot compute these new tokens. Since we assume that the token length is large enough to discourage an exhaustive attack, u cannot derive K_G^+, nor any future group key. Therefore, forward security is guaranteed. □

Theorem 2. *The recovery protocol guarantees forward security.*

Proof. Given the assumptions on the cipher strength and key length, the proof consists in showing that all group users but colluding ones can derive the next group key K_G^+ from an execution instance of the recovery protocol, and that colluding users cannot derive neither it nor any future group key.

Thanks to HANDLER RH1, every non colluding user belonging to a compromised subgroup retrieves the new group key K_G^+. Then, thanks to HANDLER RH2, every user belonging to a non compromised subgroup retrieves the new group key K_G^+. It follows that all non colluding users are successfully rekeyed.

Also, colluding users do not hold cryptographic keys used to protect rekeying messages. Then, they are not able to retrieve the new security material. Since we assume that the key length is large enough to discourage an exhaustive attack, colluding users cannot derive K_G^+, nor any future group key. Thus, forward security is guaranteed. □

Theorem 3. *HISS guarantees forward security.*

Proof. The proof descends directly from Theorem 1, in the case of a single user's leaving, and Theorem 2, in the case of colluding users' leaving. □

Theorem 4. *HISS guarantees backward security.*

Proof. First, the join protocol provides current users with new keys. Then, once current users have been rekeyed, the joining user u is provided with the new keys as well, i.e. K_G^+ and K_S^+. As a consequence, u never has knowledge of security material used before its join, and thus is not able to access old communications. Therefore, backward security is assured. □

4.3.1 On rekeying message authenticity

Authenticity of rekeying messages specified in Sections 4.2.3 and 4.2.3 must be guaranteed. Otherwise, an adversary could modify in-transit rekeying messages or inject fake ones, so completely breaking the rekeying protocol. In this section, we discuss techniques that could be used in HISS to assure the authenticity of rekeying messages. The choice of one of them is tightly related to the current application scenario, the network technology, as well as the hardware capabilities of the network devices.

Digital signatures are a typical and widely adopted solution for providing rekeying messages authentication [23][33]. However, they are quite onerous from a computation and communication standpoint [10]. The communication overhead derives from the increased size of packets due to the appending of the digital signature to the message itself. For instance, in RSA-1024, the increment of a message size is 128 bytes. Since rekeying messages are "short", a digital signature may be even larger than the information it protects. Elliptic Curve Cryptography (ECC) ameliorates this situation [28]. For instance, ECC-160 digital signature is, roughly, an order of magnitude faster than RSA-1024, and increases a message by only 40 bytes while delivering the same security level.

While digital signatures are adequate for conventional distributed applications (e.g. teleconferencing or content distribution), they may result practically unfeasible for resource-constrained computing platform such sensor nodes in WSNs. Here, even ECC-based digital signatures especially conceived for sensor nodes may result too computing and energy demanding [14][48]. In this case, authentication techniques more efficient than digital signatures have to be exploited. For instance, group rekeying schemes for WSNs such as LARK [44], S2RP [40][42], μTESLA [12], and LEAP++ [21] use *hash-chains*, an authentication mechanism deriving from Lamport's one-time password [65]. The advantage of an hash-chain is that the current element in the chain can be efficiently authenticated by computing its hash and verifying that the result is equal to the previous element in the chain. Therefore, it is sufficient to distribute the head of the chain in an authenticated way, e.g. off-line or through a predefined point-to-point authenticated channel, so that all the remaining items can be automatically and efficiently authenticated.

In HISS, hash-chains may be used to authenticate both the new group key and the new subgroup key in the leave protocol messages M1 and M2, and the new group key and the new subgroup tokens in the recovery protocol message $\mathrm{M_S}$. Therefore, the authentication of these quantities would require just the execution of a hash function. Furthermore, these quantities are "self-authenticating", so avoiding the need for a MAC or a digital signature which would increase the rekeying and recovery message size. Finally, it is worth noting that the contents of the recovery protocol message $\mathrm{M_u}$ does not require any special technique to be authenticated, neither the digital signature nor the hash-chains, because it is encrypted by means of the user key.

Using hash-chains has cons too. In fact, the previous items in the chain can be computed from the current one by repeatedly computing a hash function. This implies that when a user joins the group she becomes able to compute all previous group keys. It follows that the backward security requirements does not hold anymore. However, backward security is not so crucial for WSNs applications [7][44]. In fact, they actually deal most with monitoring and control, where integrity is a top priority. In this application domain, supporting the backward security requirement is not critical from an integrity standpoint. Actually, the adversary may only attempt to inject messages by using "past" group keys, but they are supposed to be discarded by the monitoring and control algorithm.

4.4 Performance evaluation

In this section, we evaluate the rekeying scheme performance with respect to the following factors: *communication overhead*, *storage overhead*, and *computing overhead*. In this evaluation, we abstract away from the specific design and implementation choices (e.g. the cryptographic suite or the network technology). This means that we evaluate the storage overhead and the communication overhead as the number of

information items that protocol actors have to store and broadcast. Similarly, we evaluate the computing overhead as the number of cryptographic operations performed during protocol instances, i.e. the number of encryptions, decryptions, and hash function executions. In any case we perform a worst-case analysis, that is we analyse the largest overhead implied by a given protocol execution instance.

For simplicity, but without lack of generality, we assume that the system is organized in p homogeneous subgroups containing m users each. It follows that the group G is composed of $n = p \cdot m$ users. We point out that HISS supports heterogeneous subgrouping policies, according to which different subgroups may contain different numbers of users. However, an homogeneous subgrouping policy allows us to perform an analytical performance analysis without any significant lack of generality.

While the communication and computing overhead is protocol-specific, the storage overhead is common to all protocols. For this reason, we analyse the storage overhead in a separate section (Section 4.4.1), and devote one section for the communication and computing overhead of each protocol (Sections 4.4.2–4.4.4).

In order to fix ideas, we also discuss the respective overheads of a WSNs application. Such a discussion is contextual to every overhead type and thus distributed over the four Sections 4.4.1–4.4.4. Specifically, we consider a group composed of 1024 sensor nodes (e.g. Mica2 or TmoteSky) interconnected by an IEEE 802.15.4 wireless network. Every sensor node is battery operated, and equipped with a microcontroller and a small amount of memory. Of course, this application is not exhaustive of the full range of possible applications in which HISS can be used. However, it allows us to give a concrete insight of the high scalability and practical sustainability of HISS.

4.4.1 Storage overhead

As to the storage overhead, the Key Manager KM stores the current group key K_G, all the p current subgroup keys, all the n user keys, all the n user tokens, and all the p subgroup tokens. Thus, the storage overhead for KM is $O_{s,km} = (2 \cdot p + 2 \cdot n + 1)$.

On the other hand, every user $u \in S$, stores: i) its user key K_u; ii) the current group key K_G; iii) the current subgroup key K_S; iv) the $(m-1)$ user tokens associated to its cognate users; and v) the $(p-1)$ subgroup tokens associated to all the subgroups different from S. Thus, the storage overhead for every user is $O_{s,u} = (p + m + 1)$.

Storage overhead $O_{s,km}$ at the Key Manager grows linearly with n. However, this is not a problem in practice because the Key Manager has plentiful of resources. The key point is that the HISS memory overhead is affordable at the user side. From this standpoint, it is important to notice that for $p \gg 1$ or $m \gg 1$, $O_{s,u} \simeq p + m$. Therefore, under the assumption that users are uniformly distributed in p subgroups of m members each, then the minimum storage overhead $O_{s,u}^{(\min)} = 2 \cdot \sqrt{n}$ for $p = m = \sqrt{n}$. Hereafter, we will assume this distribution of users to subgroups.

In the WSN application, the minimum storage overhead is $O_{s,u}^{(\min)} = 64$. If we assume that secrets (keys and tokens) have a size equal to 80 bits, then the storage

overhead is 640 bytes. The rationale basis of this choice is that 80 bits is the size of keys of Skipjack, a secure and efficient cipher for WSN applications [9][20][54]. If sensor nodes belong to the TmoteSky class, they are equipped with 48 Kbytes of memory [74], and thus the memory overhead is the 1.30% of the total sensor node's memory. If, instead, sensor nodes belong to the Mica2 class, they have 128 Kbytes of memory, and thus the memory overhead becomes the 0.49% of the memory. It follows that the storage overhead of HISS is practically negligible even in resource-poor devices such as sensor nodes.

4.4.2 The leave overhead

Let us consider the leave protocol (see Section 4.2.3), and a user $u \in S$ which leaves the group G. The communication overhead accounts for messages M1 and M2. Message M1 introduces a communication overhead equal to three, as it conveys the identifier of the leaving user, the next group key K_G^+, and the next subgroup key K_S^+. Message M2 introduces a communication overhead equal to two, as it conveys the identifier of the subgroup S and the next group key K_G^+ in its encrypted form. It follows that the total communication overhead is constant and equal to five, i.e. $O_c^{(l)} = 5$.

From these initial considerations, we can conclude that HISS provides a small and constant communication overhead that is independent of the group size and the adopted subgrouping strategy. This property is particularly important because it makes HISS highly scalable. Of course, in a real implementation setting, the possibility of exploiting this property strictly depends on the broadcast communication service provided by the underlying network technology.

As to the computing overhead, the worst case regards the operations performed by a user $v \in S, v \neq u$. Such a user is required to: i) compute the key encryption key KEK_u by means of *kdf()*; ii) decrypt the message M1 to retrieve K_G^+ and K_S^+; iii) update its own User Token Set UTS_v by executing $(m-2)$ hash functions, and its own Subgroup Token Set STS_S by executing $(p-1)$ hash functions. Thus, in the worst case, a user performs one decryption and $(p+m-2)$ hash function executions.

On the other hand, the Key Manager has to: i) compute the new group key K_G^+ and the new subgroup key K_S^+, as well as the two key encryption keys KEK_u and KEK_S, by means of *kdf()*; ii) encrypt messages M1 and M2; and, finally, iii) update its own User Token Set UTS by executing $(m-1)$ hash functions, and its own Subgroup Token Set STS by executing p hash functions. Thus, the Key Manager has to perform 2 encryptions and $(p+m+3)$ hash function executions.

As it turns out, the computing overheads of the Key Manager and users have the same order of magnitude. Once again, since the Key Manager is plentiful of resources, the practical viability of HISS from the computing standpoint depends on the users. In the considered WSN application, users are required to perform at most 1 decryption and 62 hash function executions, which is practically affordable for sensor nodes.

4.4.3 The recovery overhead

Let us consider the recovery protocol (see Section 4.2.3), with C compromised subgroups and $(p-C)$ non-compromised subgroups. For the sake of simplicity but without any lack of generality, let us assume that every compromised subgroup contains c colluding captured users. It follows that the overall captured users are $(c \cdot C)$, whereas non-captured users are $(n - c \cdot C)$.

The communication overhead accounts for messages of type $\mathrm{M_u}$ and $\mathrm{M_S}$. For every compromised subgroup, the Key Manager sends a message of type $\mathrm{M_u}$ to every non-captured node in the subgroup. The size $\|\mathrm{M_u}\|$ of a message of type $\mathrm{M_u}$ is $\|\mathrm{M_u}\| = (p + m - c)$, as the message conveys the new group key K_G^+, the new subgroup key K_S^+, the new User Token Set UTS_u, and the new Subgroup Token Set STS_S. As the number of compromised subgroups is C, and the number of non-captured users in every compromised subgroup is $(m - c)$, then the total overhead due to messages of type $\mathrm{M_u}$ is $C \cdot (m - c) \cdot (p + m - c)$.

A message of type $\mathrm{M_S}$ is broadcast to every non-compromised subgroup $S \in \mathcal{U}$. Thus, the number of these messages is $(p - C)$. The size $\|\mathrm{M_S}\|$ of a message of type $\mathrm{M_S}$ is $\|\mathrm{M_S}\| = p$, as it conveys the new group key K_G^+ and the new Subgroup Token Set STS_S. Therefore, $\mathrm{M_S}$ messages introduce a total overhead equal to $p \cdot (p - C)$. Consequently, the total communication overhead of the recovery protocol is equal to $O_c^{(r)} = [C \cdot (m - c) \cdot (p + m - c)] + p \cdot (p - C)$.

If we reasonably assume that i) each subgroup includes a non-negligible number of members, i.e. $m \gg 1$; ii) a few users per subgroup are captured, i.e. $m \gg c$; iii) a few subgroups are compromised, i.e. $p \gg C$; and iv) $p = m = \sqrt{n}$, for storage optimisation, then M_u, M_S, and $O_c^{(r)}$ can be approximated as follows: $\|\mathrm{M_u}\| \simeq 2 \cdot \sqrt{n}$, $\|\mathrm{M_S}\| \simeq \sqrt{n}$, and $O_c^{(r)} \simeq (2 \cdot C + 1) \cdot n$. As it turns out, the total communication overhead linearly grows with C, so displaying a *smooth degradation* behavior.

The total communication overhead also grows linearly in n. However, an execution of the recovery protocol has to be considered an exceptional event with respect to an execution of the leave protocol, that is instead a normal event. So, according to the well-known Lampson's recommendations for computer systems design [19], while the leave protocol must be efficient and scalable, the recovery protocol must be able to make some progress. However, it is crucial that the recovery protocol is sustainable. In the rest of this section, we argue that this is indeed the case.

In the WSN application, we have $\|\mathrm{M_u}\| = 640$ bytes, $\|\mathrm{M_S}\| = 320$ bytes and $O_c^{(r)} = 30$ Kbytes. Let us consider the IEEE 802.15.4 communication protocol [52], where unsecured frames have a payload whose size can be up to 102 bytes. Thus, every message $\mathrm{M_u}$ requires 7 frames, whereas the transmission of the whole $O_c^{(r)}$ overhead requires 302 frames. In an implementation of IEEE 802.15.4, the effective data rate (i.e. excluding headers, CRCs, and control packets) is about 8.4 Kbps (out of 250 Kbps), thus the transmission of every message $\mathrm{M_u}$ requires about 610 ms (per-hop), whereas the transmission of the whole $O_c^{(r)}$ overhead requires about 29.25 s

(per-hop). It follows that also in this case the communication overhead is sustainable. However, it is worthwhile to notice that IEEE 802.15.4 is conducive to provide better performance. For instance, Latré *et al.* showed that a throughput of about 140 Kbps can be achieved in IEEE 802.15.4, even if Acknowlegment frames are trasmitted [16]. In such a case, the trasmission of a message $\mathrm{M_u}$ requires about 36.57 ms (per-hop), whereas the transmission of the whole $O_c^{(r)}$ overhead requires 1.75 s (per-hop).

As to the computing overhead, each user performs only one decryption, in order to decrypt either a message $\mathrm{M_u}$ or a message $\mathrm{M_S}$. On the other hand, the Key Manager has to: i) compute K_G^+, $(p-C)$ new subgroup keys, $C \cdot (m-c)$ new user tokens, and p new subgroup tokens by means of *kdf*; ii) encrypt $C \cdot (m-c)$ messages $\mathrm{M_u}$ and $(p-C)$ messages $\mathrm{M_S}$. Thus, the Key Manager has to perform $p + C \cdot (m-c-1)$ encryptions and $2 \cdot p + C \cdot (m-c-1) + 1$ hash functions.

On the other hand, the computing overhead at the Key Manager side is generally not a problem, because the Key manager may be implemented on a high-performance server platform. Therefore, the actual sustainability from the computing standpoint practically depends on the user side.

Particular attention must be paid to the WSN application scenario, where users are sensor nodes with constrained computing capabilities. In such a case, messages $\mathrm{M_u}$ and $\mathrm{M_S}$ are 640 bytes and 320 bytes in size, respectively. On a Mica2 node, the performance of a software-version of Skipjack is 25 μs per encrypted/decrypted byte [20], whereas, on a TmoteSky node, it is 77 μs [44]. Hence, decrypting a message $\mathrm{M_u}$ or a message M_S takes 16 ms and 8 ms, respectively, on a Mica2 node, and 49.28 ms and 24.64 ms, respectively, on a TmoteSky node. This makes the HISS recovery protocol affordable from the computing standpoint too.

4.4.4 The join overhead

Let us consider the join protocol (Section 4.2.3), and a user u which wants to join the group G and become a member of the subgroup S. An execution instance of the protocol requires a constant number of rekeying messages, namely i) a broadcast message providing the new group key K_G^+; and ii) a broadcast message providing the new subgroup key K_S^+ and the user token t_u to the members of S. Thus, the total communication overhead amounts to three, i.e. $O_c^{(j)} = 3$.

As to the computing overhead, the worst case regards the operations performed by the members of the subgroup S. Each one of them is required to perform two decryptions, in order to retrieve K_G^+, and K_S^+ and t_u from messages M1 and M2, respectively. Conversely, the Key Manager has to: i) compute the new group key K_G^+, the new subgroup key K_S^+, and the user token t_u; and ii) encrypt messages M1 and M2. Thus, the Key Manager has to perform 2 encryptions and 3 hash function executions.

4.5 Collusion management

So far, we have shown that if two captured users belonging to the same subgroup collude, then the whole subgroup is compromised. However, we have also shown that recovering a compromised subgroup requires only to totally recover such a subgroup rather than the whole group. The remaining uncompromised subgroups can be then efficiently rekeyed by means of a single broadcast message each.

We argue that a proper allocation of users to subgroups can make a successful collusion attack *practically* unfeasible. As we consider a form of practical security, we cannot abstract away from the application scenario and the hypothesized adversary strength. So, given an application scenario, we have first to assess the adversary strength, and then devise an allocation strategy that is consistent with the application constraints and requirements, and makes a collusion attack practically unfeasible for the alleged adversary. This is a common process in security management [26].

In any case, the chosen allocation strategy has to satisfy the general principle that, for an adversary, it must be practically unfeasible to i) determine the subgroups membership, or ii) compromise two or more users belonging to the same subgroup.

In the former case, the only strategy for the adversary consists in exhaustively attacking users in the attempt to compromise two users belonging to the same subgroup. The system is practically secure if the resources required to carry out this exhaustive attack strategy exceeds the resources of the hypothesized adversary. For instance, if we consider the Internet and its magnitude, determining subgroups and their membership through traffic analysis is a practically unfeasible task.

In the latter case, subgroups membership is known to the adversary, who can thus target the attack to well defined users. Therefore, the system is secure if the resources required to attack, physically or logically, two or more users belonging to the same subgroup exceed the resources of the adversary. In this case, off-the-shelf protection measures and security engineering best-practices can be employed [88][105].

In order to fix ideas, we refer again to the WSN application scenario that so far have constituted the leading example (see Section 4.4). In particular, we discuss a possible instantiation of the allocation strategy principle, that depends on the specific application requirements and constraints as well as the related hypothesized adversary. We consider an homogeneous allocation of users to subgroups, with p subgroups containing m members each, and a total group size $n = p \cdot m$. Once again, it is arguable that, although not exhaustive, it however constitutes a meaningful example to get a concrete insight of the HISS flexibility in collusion management.

4.5.1 Wireless Sensor Networks applications

WSNs are often employed in monitoring, surveillance, and control of buildings, plants, or critical infrastructures. In these application scenarios, we can assume that an adversary is able to physically break into a given area, and capture “neighbouring” sensor nodes, possibly all, deployed in that area. Once a sensor node has been captured,

all secrets it holds become known to the adversary. Thus, she can gather secrets possibly from all sensor nodes in that area, and possibly collect enough information to gain access to the group key. It has been experimentally shown that an equipped and experienced adversary is able to obtain copies of all memory and data of, for instance, a Mica2 mote in tens of seconds, or minutes, after a node is captured [22].

On the other hand, due to the reduced amount of services provided by sensor nodes, it is believed that a WSN presents a number of vulnerabilities which is smaller with respect to computers typically connected to the Internet. Since there are less functionalities and less complex code, there are less software bugs too. Also, as sensor nodes are resource-constrained, programmers spend more time per line of code in sensor network applications, than in applications for regular computer networks [7].

Thus, it is the size of the area the adversary can physically compromise that gives a practical measure of the adversary's strength. Such a "critical" value is typically defined at pre-deployment after a threat analysis [26]. It follows that the problem that has to be solved consists in devising an allocation and deployment policy such that any area not larger than the critical size contains sensor nodes belonging to different subgroups with high probability.

It is well-known that devising an optimal allocation policy is in general an NP-hard problem [63][71]. Therefore, we have to resort to heuristics for the assignment of users to subgroups that reduce, or even practically eliminate, the probability of successful collusion attacks. Such heuristics depend on both the adversary model and application-specific requirements and constraints. In this section, we discuss a real case-study, the "Highly Automated Airfield" (HAA) reference application of the PLANET European integrated project [82].

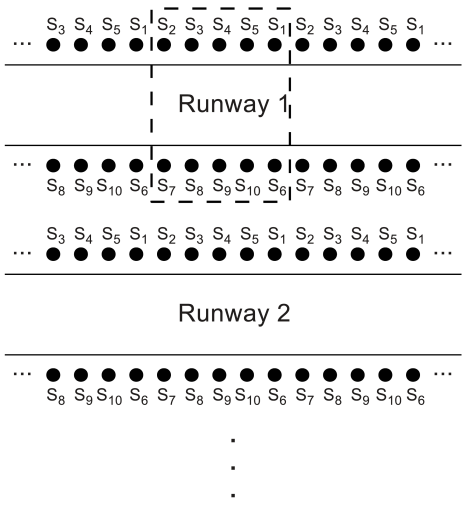


Figure 4.3. The HAA application: an allocation policy for $w = 5$. Black circles represent sensor nodes, while the dashed rectangle depicts the largest area that the adversary can compromise.

The HAA application provides the support for automated airfield operations involving cooperating unmanned air vehicles (UAVs). In particular, by means of a WSN, the HAA application intends to monitor the runways of an airport in order to permit automated take-offs and landings of UAVs. The HAA application considers a WSN composed of n sensor nodes, evenly distributed on both sides of every runway to monitor both the runway occupancy and the presence of obstacles. As the airport is provided with a physical land surveillance system, we assume that an adversary can compromise a portion L of a runway, possibly spanning both sides (see Figure 4.3), and thus capture the w "neighbouring" sensors deployed therein. The adversary cannot move from one runway to another without being caught.

In order to avoid collusion attacks, we organize sensor nodes into $p = 2 \cdot w$ different subgroups with $m = n/(2 \cdot w)$ members each, and cyclically allocate sensor nodes to subgroups. Figure 4.3 shows an example for $w = 5$. Since the adversary is not able to compromise a number of sensor nodes larger than w, nor to move from one runway to another, then she cannot successfully perform a collusion attack. It is worth noting that the application-specific constraints and requirements have allowed us to reduce the allocation problem to a single-dimension problem, and thus devise an effective allocation strategy that prevents collusion attack under the considered threat model.

w	Memory (bytes)	TmoteSky (%)	Mica2 (%)
32	800	1.66	0.62
64	1360	2.83	1.06
128	2600	5.41	2.03
256	5140	10.70	4.01

Table 4.1. Highly Automated Airfield: storage overhead.

Table 4.1 reports the storage overhead in the TmoteSky and Mica2 platforms, for different values of w. The column "Memory" reports the amount of storage necessary for the sensor node to store the HISS keys and tokens (see Section 4.4.1). The columns "TmoteSky" and "Mica2" specify, respectively, the percentual impact of this amount of storage on the total memory available on the platform. It follows that the proposed allocation policy has a practically affordable storage overhead. Specifically, in the most extreme case of tolerating the capture of $w = 256$ sensor nodes, the overhead is around 10% on the TmoteSky platform and 4% on the Mica2 platform.

5

Programming secure Wireless Sensor Networks

Wireless Sensor Networks (WSNs) have been used chiefly for scientific purposes, where an adversary has little incentive to attack sensors [7]. However, WSNs are now employed also in Cooperating Objects Systems (COS) where mobile physical agents share the same environment to fulfill their tasks, either in group or in isolation [30][82]. Such agents not only sense the environment, but also act on it. COS 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. Same considerations hold for WSNs in Critical Infrastructures [64][66].

However, sensor nodes are typically resource constrained devices deployed in unattended, possibly hostile environments. Given the nature of WSNs, it is an easy task for an adversary to eavesdrop messages and alter or inject fake ones. It follows that secure communication is vital to assure messages confidentiality, integrity, and authenticity. Many solutions have been devised for WSNs security, including [25] for secure communication, [23][45][57][104][113] for key management, and [80][90] for secure code dissemination. Particular attention has been paid to component-based security architectures tailored to WSNs [43][75].

In this chapter, we present *STaR* [87], 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. This makes it easily portable on different hardware [31][74] and network stacks [52][116]. Modularity allows for loading/unloading STaR modules to match security requirements, add new features, or extend existing ones.

STaR allows for protecting multiple traffic flows at the same time, according to different security policies. Also, it 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 characteristics allow for reusing application components in scenarios where security becomes relevant. Also, STaR allows unskilled people 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.

We evaluate our STaR implementation [86] for TinyOS [100] on Tmote Sky motes [74], showing it is affordable as to memory occupancy, communication overhead, and energy consumption. STaR features a generic architecture, and can be implemented for other hardware platforms and WSNs operating systems.

5.1 STaR architecture

STaR allows for securing multiple *traffic flows* according to different *security policies*. Possible security policies include packet encryption, packet authentication, or both of them. The choice of appropriate security policies is related to the specific application and threat model, and is out of the scope of this dissertation.

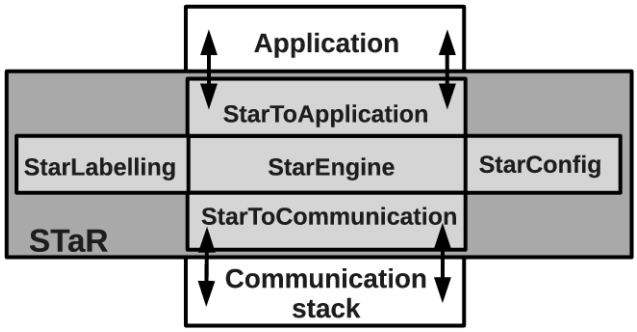


Figure 5.1. STaR component overview.

As shown in Figure 5.1, the STaR component stays between the application and the communication stack. STaR intercepts both incoming and outgoing traffic, segments it into flows, and secures them according to the corresponding security policies. STaR assures *reconfigurability* by allowing users to dynamically change security policies, provides *transparency* of security with respect to the application, and exports the same interface as that of the underlying communication stack to the application.

The *StarToApplication* component provides the application with the same communication interface exported by the communication stack. The *StarToCommunication* component connects STaR to the underlying communication stack. The *StarLabelling* component classifies packets into *traffic flows*, and determines the associated label. The *StarConfig* component allows users to enable/disable security policies, and change their association to traffic flows, providing reconfigurability at runtime.

Figure 5.2. Example of packet processed by STaR.

The *StarEngine* component actually processes packets, according to the security policy associated to the traffic flow they belong to. Distribution and revocation of necessary security material are left to other system components, which are not part of STaR and are out of the scope of this dissertation. Figure 5.2 shows a packet processed by STaR, with the *Label* field prepended to the packet payload.

STaR modularity eases the porting of STaR onto different communication stacks. Actually, different communication stacks require 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.

5.2 STaR security services

As described in Section 5.1, 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, i.e. 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. In order to manage the *SPT* at runtime, STaR provides a set of configuration functions, which we describe in Section 5.2.2.

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 implemen-

tation provided by STaR. The *EntryPoint* field contains a reference to the code section which implements the policy (e.g. a C++ function pointer).

5.2.1 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.

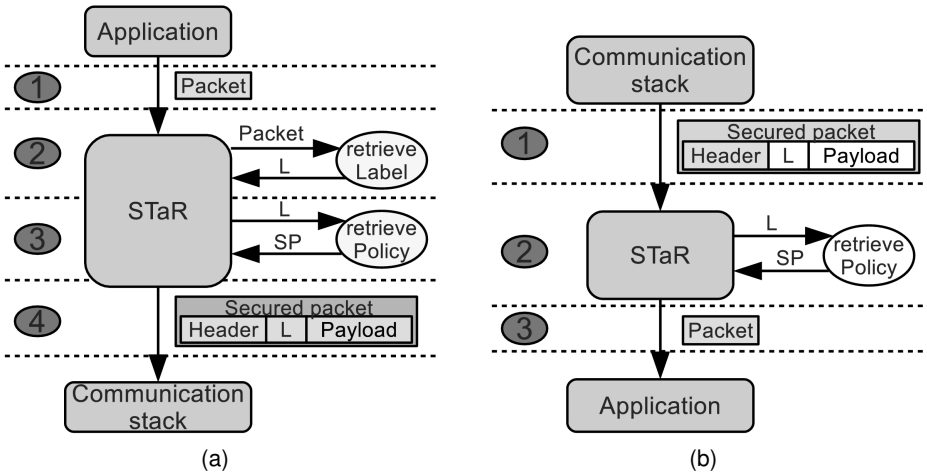


Figure 5.3. A) Outgoing packet processing and B) incoming packet processing.

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 5.3-A 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 field, which can then be removed. Then, STaR retrieves the security policy SP associated to label L, through the *retrievePolicy* function, and unsecures packet P, according to SP. Finally, STaR provides the unsecured packet to the application, that receives it through the *receive* function. Figure 5.3-B shows an incoming packet processed by STaR.

5.2.2 STaR configuration services

STaR allows users to change security settings at runtime, and provides a specific *configuration interface* aimed at changing how security policies are used, and their association to traffic flows. In the following, we describe the configuration functions.

```
void enablePolicy(label);
```

Set to TRUE the *Active* field of the *label SPT* entry.

```
void disablePolicy(label);
```

Set to FALSE the *Active* field of the *label SPT* entry.

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

Write *newPolicy* in the *PolicyID* field of the *SPT* entry related to *label*. The *Active* field remains unchanged.

5.3 STaR TinyOS implementation

We implemented the STaR component [86] on TinyOS 2.1.1 [100], for the Tmote Sky motes [74] and the CC2420 chipset [99]. Also, we implemented the *Skipjack* encryption module [101] and the *SHA-1* module for integrity hashing [32].

5.3.1 STaR memory footprint

ROM memory available on Tmote Sky motes is 48 KB, and may be a severe constraint while dealing with complex modules like those composing STaR. In order to evaluate memory consumption on Tmote Sky motes, we considered the TinyOS image size wiring the STaR submodules separately.

S is the image size in bytes of the original TinyOS stack and the *RadioCountToLeds* application, which is one of the provided demo applications.

$\hat{\mathrm{C}} = \mathrm{S} + \mathrm{C}$ is the image size in bytes of the original TinyOS stack and the *RadioCountToLeds* application (S), wired to the *StarConfig* module (C). $\mathrm{C} = \hat{\mathrm{C}} - \mathrm{S}$ is the memory occupancy of the *StarConfig* module.

$\hat{\mathrm{E}} = \mathrm{S} + \mathrm{C} + \mathrm{E}$ is the image size in bytes of the original TinyOS stack and the *RadioCountToLeds* application (S), wired to the *StarConfig* module (C) and the *Skipjack* submodule of the *StarEngine* module (E). $\mathrm{E} = \hat{\mathrm{E}} - \hat{\mathrm{C}}$ is the memory occupancy of the *Skipjack* submodule of the *StarEngine* module.

$\hat{\mathrm{A}} = \mathrm{S} + \mathrm{C} + \mathrm{A}$ is the image size in bytes of the original TinyOS stack and the *RadioCountToLeds* application (S), wired to the *StarConfig* module (C) and the *SHA-1* submodule of the *StarEngine* module (A). $\mathrm{A} = \hat{\mathrm{A}} - \hat{\mathrm{C}}$ is the memory occupancy of the *SHA-1* submodule of the *StarEngine* module.

	Memory occupancy (B)	**Memory occupancy (%)**
Application and TinyOS stack	13372	27.86
StarConfig	854	1.78
StarEngine (Skipjack)	2046	4.26
StarEngine (SHA-1)	3900	8.12
Available memory	27828	57.98

Table 5.1. Detailed memory occupancy.

Table 5.1 provides information on memory occupancy, and shows the percentages of Tmote Sky ROM occupied by the original application and STaR (sub)modules. If we sum the contributions of the *StarConfig*, *Skipjack* and *SHA-1* modules, we observe our STaR implementation totally requires the 14.16% of the overall memory available on a Tmote Sky mote. Since the application together with the TinyOS stack requires the 27.85% of the available memory, the 57.99% of 48 KB is available for other uses. Thus, memory required by STaR is reasonable with respect to the available memory.

5.3.2 STaR performance evaluation

In our analysis, we assumed STaR operates on top of the 2.4 GHz physical layer, with a 250 Kb/s bit rate [99]. We modeled the impact of security considering i) network performance degradation due to security processing and extra trasmission overhead,

and ii) the extra energy consumption, due to extra processing and transmissions. We evaluated the security processing overhead by means of experiments. The extra transmission overhead has been computed considering the bit rate and the packet size. Instead, energy consumption has been evaluated analytically.

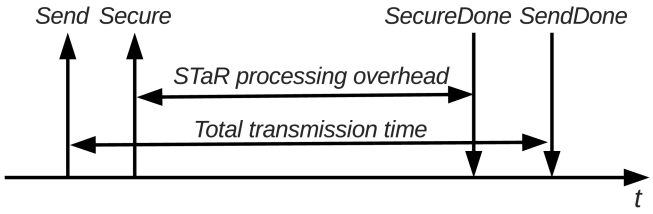


Figure 5.4. Send and Secure events nesting.

Figure 5.4 shows events which take place when we transmit a secured packet with STaR, according to the TinyOS Send/SendDone schema. The *STaR processing overhead* time interval is the extra time required to process the packet according to the chosen security policy. This time has been evaluated experimentally.

Policy	$\mathbf{d_{proc}}$ (μs)	Standard deviation (μs)
NONE	142	0
ENC	1239.70	1.96
HASH	32853.50	2.53
ENC + AUTH	33948.65	3.01

Table 5.2. STaR $\mathrm{d_{proc}}$ contributions overview.

In our experiments, we observed one sender device at a time transmitting secured packets whose payload is 8 bytes in size. In order to increase the accuracy of our results, we performed 10 repetitions of 20 transmissions for each experiment. Results shown in Table 5.2 are averaged over all different repetitions. We also report the standard deviation we derived from the independent replication method.

Considerable delays are due to the standard encryption and authentication algorithms, while the actual STaR contribution to the processing delay is just 142 μs. This is the additional delay required to add the label field, which is negligible if compared to the one introduced by standard cryptographic computations, such as those performed by *Skipjack* and *SHA-1*.

The transmission overhead has been evaluated analytically, considering a 250 Kb/s bit rate [99]. Specifically, we have considered the time required to transmit the additional bytes added by STaR, according to the specific security policy. The original application packet size, including the header and the *Cyclic Redundancy Check*

(*CRC*) is 21 bytes. The time $\mathrm{d_{tx}}$ required to transmit the original application packet is the ratio between the packet size in bits and the bit-rate: $\mathrm{d_{tx}} = \frac{21 \cdot 8}{0.250} = 672\ \mu\mathrm{s}$.

Policy	$\mathbf{d_{tx}}$ (μs)	Increase (%)
NONE	32	4.76
ENC	32	4.76
HASH	672	100
ENC + AUTH	672	100

Table 5.3. STaR $\mathrm{d_{tx}}$ contributions overview.

Table 5.3 provides an overview of the transmission overhead, considering different security policies. Considerable delays of the *HASH* and *ENC + AUTH* policies are due to the standard *SHA-1* hashing output size, which is 20 bytes long. In fact, the actual STaR contribution to the transmission delay is just 32 μs, that is the time required to transmit the one byte label field added to the original packet. We believe this delay is affordable, since it is due to the increase of just one byte of the original packet size.

We expressed energy consumption contributions as $\mathcal{E}_\mathrm{i} = \mathcal{P}_\mathrm{i} \times \mathrm{d_i}$. Let $\mathrm{d_i}$ be the delay due to the considered operation i. $\mathcal{P}_\mathrm{i} = \mathrm{V_i} \times \mathrm{I_i}$ is the single power contribution, i.e. the product between voltage and current of the MSP430 and CC2420 components.

Policy	Processing $\mathcal{P}_\mathrm{proc} = 1.08\ \mathrm{mW}$		Transmission $\mathcal{P}_\mathrm{tx} = 31.32\ \mathrm{mW}$	
	$\mathrm{d_{proc}}(\mu\mathrm{s})$	$\mathcal{E}_\mathrm{proc}(\mathrm{nJ})$	$\mathrm{d_{tx}}(\mu\mathrm{s})$	$\mathcal{E}_\mathrm{tx}(\mathrm{nJ})$
NONE	142	153.4	32	1002.2
ENC	1239.7	1338.9	32	1002.2
HASH	32853.5	35481.8	672	21047.0
ENC + AUTH	33948.7	36664.6	672	21047.0

Table 5.4. STaR energy consumption contributions.

Table 5.4 provides an overview of such contributions. Considerable increases in per packet energy consumption are due to the standard encryption and authentication algorithms, while the actual STaR contribution to energy consumption is the one reported in the *NONE* policy entry: $\mathcal{E}_\mathrm{proc} + \mathcal{E}_\mathrm{tx} = 1155.6\ \mathrm{nJ}$ that is the energy consumed to add the label field to the original packet and transmit it. We believe that the extra energy consumption is affordable, if compared to contributions due to standard security mechanisms as *SHA-1*.

6

Simulative evaluation of security attacks

Wireless Sensor Networks (WSNs) are a widely adopted technology in several fields of application. Their consolidated success, both in the academic and industrial world, is mainly due to standards availability, low cost of sensor nodes, and world-wide communities support. Their uses include home automation, plant monitoring, military surveillance, disaster recovery situations, environment monitoring, and many others.

A WSN may be affected by a great number of security threats and attacks. WSNs are subject to *logical attacks* (aka *cyber attacks*), as an adversary simply equipped with a radio receiver-transmitter can easily eavesdrop as well as inject messages. Furthermore, WSNs are also subject to *physical attacks*. Actually, WSNs are often deployed in environments that are open, unattended, and possibly hostile. This allows an adversary to physically attack sensor nodes in order to reprogram, misplace, or, even, break them. All these attacks may lead to unreliable data collection, inaccuracy in controlling processes, or even safety problems.

There are lot of possible attacks against WSNs [49][89][98]. They can have different objectives, be performed at different levels, and result in different effects. Physical attacks comprise node capture [81], and tampering with sensor nodes [8]. On the other hand, logical attacks include the HELLO flooding attack [24], the Sybil attack [59], the sinkhole attack [24], the wormhole attack [106], the blackhole attack [17], and the Distributed Denial of Service (DDoS) attack [103].

However, providing a security countermeasure for every possible attack would result in a huge security overhead, which is difficult to afford and would overwhelm the scarce resources available on sensor nodes [84][85]. Thus, it is vital to properly rank security attacks, in order to evaluate their severity, state protection priorities, and select appropriate countermeasures. That is, it would be useful to quantitatively evaluate the effects of attacks, in order to assign them a priority, and select adequate countermeasures. Most of the times, this information becomes available when it is too late, i.e. once attacks have already occurred. Instead, it would be better to have it even before network deployment takes place.

In this chapter, we present ASF, our attack simulation framework for WSNs [46]. Thanks to ASF, users can describe attacks, reproduce their effects, and quantitatively evaluate their impact on the overall functionality and performance. ASF helps users to evaluate the severity of an attack, and therefore constitutes a valuable tool for the attack ranking process. We show that the architecture of ASF is general and independent of the underlying network simulator.

Also, we present an early prototype of ASF built on top of Castalia [2], the popular WSNs simulator based on the OMNeT++ platform [3]. Then, we show the capabilities of ASF by considering a realistic case study and analysing four different attacks. Specifically, we compare the effects of these attacks on system functionality and performance with respect to the case when the system is attack-free.

6.1 Attack simulation background

Simulative analysis can provide a quantitative idea of the effects of attacks. However, little work has been done on attack simulation so far. In fact, most of the adopted approaches focus on network simulation and performance evaluation. Also, they consider analytical models or algorithms aimed at *detecting* specific attacks upon their occurrence. Then, simulation is typically used to validate such models.

For instance, [112] describes a distributed wormhole detection algorithm, and shows simulation results in order to prove its low false toleration and false detection rates. In [91], Kaplantzis *et al.* propose an intrusion detection scheme to detect blackhole and selective forwarding attacks. Also, simulative results are presented to validate such scheme. In [50], the authors propose a scheme based on weighted-trust evaluation, aimed at detecting malicious nodes, and verify its correctness and efficiency by simulations. Bonaci *et al.* consider physical node capture attacks, develop a network response strategy, and validate it by means of simulations [97].

In [108], Wang and Bagrodia present *SenSec*, a framework for security evaluation in WSNs. SenSec allows for simulating the occurrence of attacks against the network, by injecting events into real application simulators. They test SenSec on *TiQ* [107], a framework for executing TinyOS applications [100], and use it to evaluate the impact of a number of attacks against routing and time synchronization protocols.

6.2 Attack simulation framework

The ASF framework provides: i) an *Attack Specification Language* to specify attacks; ii) an *Attack Database* to store attack descriptions; iii) an *Attack Compiler* to convert attack descriptions into XML configuration files; and iv) an *Attack Simulator* to evaluate effects of attacks. Figure 6.1 shows an overview of the framework architecture.

First, the user specifies attacks to be evaluated by means of the *Attack Specification Language* (*ASL*), and possibly stores attacks descriptions in the *Attack Database*.

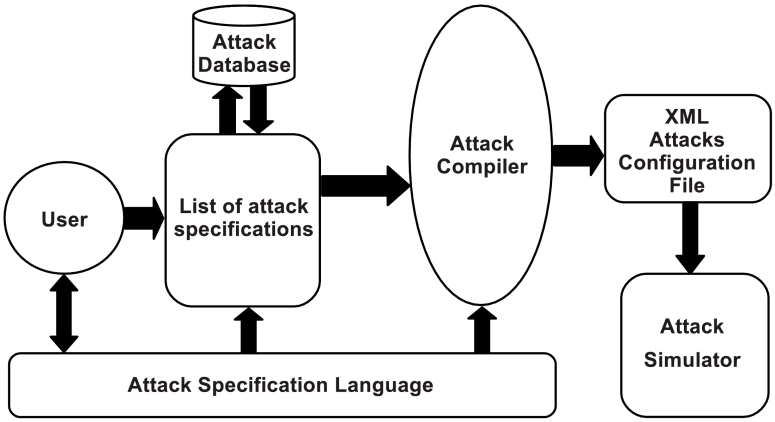


Figure 6.1. Overview of the ASF framework architecture.

Then, the *Attack Compiler* (*AC*) takes such specifications as input, and converts them into an XML *Attacks Configuration File* (*ACF*). The user must specify the name of the specific simulator to be used. By doing so, XML statements in the *ACF* can be correctly parsed and acquired by the adopted simulator. Finally, the *Attack Simulator* takes the *ACF* as input, in order to reproduce attacks and evaluate their effects.

6.2.1 Attack specification

The *Attack Specification Language* (*ASL*) is a collection of *primitives* which allow users to specify attacks to be evaluated. From our framework standpoint, we define an attack as a sequence of *events*, which takes place atomically. That is, the user specifies the sequence of events that compose an attack by means of *ASL* primitives.

We consider two sets of primitives. *Node primitives* allow for altering node behavior, thus performing physical attacks. Instead, *packet primitives* allow for performing a number of actions on network packets, thus carrying on logical attacks. The following two *node primitives* are available.

destroy*(nodeID, t)*;
At time *t*, remove node *nodeID* from the network.

move*(nodeID, t, x, y, z)*;
At time *t*, move node *nodeID* to position (*x*;*y*;*z*).

Also, the following six *packet primitives* are available.

drop*(packet)*;
Discard the packet *packet*.

create*(packet, field, content)*;
Create a new packet *packet*, and fill its field *field* with *content*.

clone*(srcPacket, dstPacket)*;
Create a new packet *dstPacket* as a perfect copy of *srcPacket*.

change*(packet, field, newContent)*;
Write *newContent* into the field *field* of packet *packet*.

retrieve*(packet, field, content)*;
Retrieve the content of the field *field* of packet *packet*, and write it into *content*.

put*(packet, dstNodes, direction, delay)*;
After *delay* milliseconds, put packet *packet* either in the transmission or reception buffer of all nodes in the *dstNodes* list. Possible values for *direction* are TX and RX, which refer to the transmission or reception buffer, respectively.

The *ASL* allows for defining three kinds of attacks, i.e. *physical attacks*, *conditional attacks*, and *unconditional attacks*. In the following, we describe how to specify them.

Physical attacks. They consist in a single event, modeled by either the *destroy()* or *move()* node primitive. As described above, such primitives require to specify the time t at which the event takes place, and the node affected by the attack.

Conditional attacks. This class of attacks considers a number of nodes that intercept packets to be examined, and, potentially, manipulated or even discarded. The user must specify the time t starting from which the attack takes place, and the list of network nodes that perform the attack.

Events occurrence may depend on the evaluation of a conditional statement, namely a *packet filter*. We define a packet filter as a set of simple boolean conditions $c_1, c_2, ..., c_N$ joined by logic operators, i.e. *AND*, *OR*, and *NOT*.

What follows is an example of conditional attack specification. Starting from time 200 s, nodes 1, 2, and 7 intercept all packets travelling through their communication stack. Then, such nodes check if each intercepted packet satisfies the packet filter conditions. In case of a positive match, the attack is actually reproduced, i.e. the specified list of events is executed.

```
from t = 200; nodes = "1 2 7" do {
  if(<packet filter>)
    <List of events>
}
```

Unconditional attacks. Unlike conditional attacks, these attacks are not related to interception of packets by network nodes. However, new packets can be created, and possibly cloned, in order to be naughtly injected into the network. The user must

specify the time t starting from which the attack takes place, and the occurrence frequency f according to which the attack has to be repeatedly reproduced over time. For this class of attacks, events occurrence does not depend on the evaluation of conditional statements.

What follows is an example of unconditional attack specification. Starting from time 200 s, the attack occurs repeatedly every 10 s, i.e. the specified list of events is executed.

```
from t = 200; every f = 10 do {
  <List of events>
}
```

In the following, we consider different kinds of attacks, and specify them by means of *ASL* primitives. For the sake of simplicity, we refer to packet fields with a simplified and extremely general notation. However, in a more general case, the user must be aware of which specific network protocols are in use. That is, for each communication layer, she must be aware of packet header structures and fields.

In such a general case, the user relies on a *packet.layer.field* notation, in order to access the field *field* of packet *packet* in the header of layer *layer*. Being available this information, it is possible to access specific header fields, in order to check and alter their content. As a practical example, the OMNeT++ platform [3] and the WSNs simulator Castalia [2] provide a set of objects, namely *descriptors*, aimed at handling packets of a given communication layer and accessing their header fields.

Node removal. Nodes 5 and 10 are removed from the network, at time 200 s and 500 s, respectively.

```
destroy(5,200);
destroy(10,500);
```

Node misplacement. Node 10 is moved to position $(80; 10; 0)$ at time 200 s. Similarly, node 11 is moved to position $(60; 10; 0)$ at time 200 s.

```
move(10,200,80,10,0);
move(11,200,60,10,0);
```

Node reprogram. Starting from time 400 s, nodes 5 and 7 replace the original payload of application data packets sent by node 10 with the minimum possible value. Then, data packets are regularly sent to their scheduled destination.

```
from t = 400; nodes = "5 7" do {
 if(packet.APP.source==10 &&
    packet.APP.type==DATA)
      change(packet,APP.payload,MIN);
}
```

Wormhole attack. Starting from time 200 s, MAC data packets sent by node 3 are intercepted. Then, they are provided to distant nodes 15, 17, and 18, which receive

them in their reception buffer after 100 milliseconds. This is basically a wormhole attack, where a subset of packets are captured from a given portion of the network, and forwarded to a number of distant nodes through a low-latency channel.

```
dstList={15,17,18};
from t = 200; nodes = "*" do {
 if(packet.MAC.source==3 &&
    packet.MAC.type==DATA)
      put(packet,dstList,RX,100);
}
```

As already said, attacks specifications can be stored into the *Attack Database*. Of course, the user can retrieve them subsequently, by querying the *Attack Database*. Querying parameters include the lapse of time during which attacks are supposed to be performed, or a list of involved nodes.

Thanks to the *Attack Database*, the user does not have to necessarily define new attacks from scratch. Instead, she can rely on stored specifications, or modify their behavior and severity for different network and application scenarios.

6.2.2 ASF network architecture

As depicted in Figure 6.2, ASF considers every node as composed by a generic *Sensing & Application* module, a *Communication stack* module, and a *Local Filter* module.

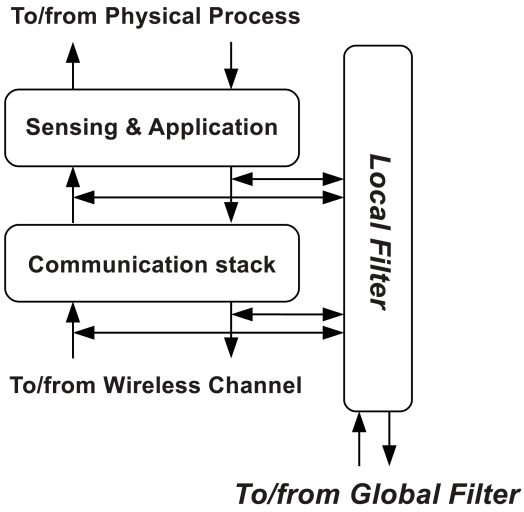


Figure 6.2. Sensor node architecture in ASF.

The Sensing & Application module may be composed of different submodules, which model the actual node application as well as physical sensing processes. The

Communication stack module may include an arbitrarily complex combination of communication layers, e.g. Routing and MAC. Finally, the Local Filter module intercepts all packets travelling through the communication stack. Thus, it is able to collect packets that the node application has transmitted or is about to receive.

The Local Filter can inspect and alter packet content, add new packets to be transmitted, or even discard them. Also, it can alter the node behavior at different layers, change the node position in space, or even remove the node from the network.

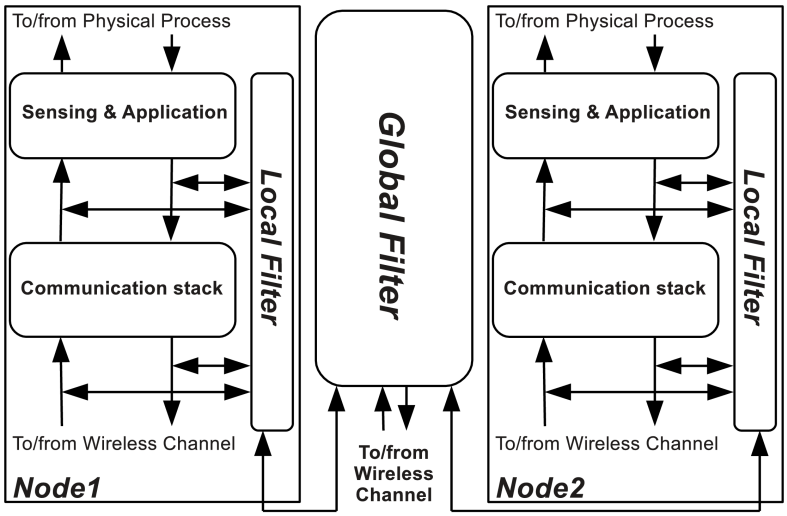


Figure 6.3. Overview of ASF network architecture.

Furthermore, one single Global Filter module is connected with every Local Filter module. That is, Local Filters of different nodes can communicate and cooperate with each other through the Global Filter module, in order to reproduce and evaluate more complex attacks, e.g. a wormhole attack. Figure 6.3 shows the overall ASF network architecture, in the presence of two sensor nodes, i.e. *Node1* and *Node2*.

6.2.3 Attack reproduction

An *Attacks Configuration File* (*ACF*) is divided into three sections. The first section regards physical attacks, and contains the list of *node primitives* to be performed. We recall that each *node primitive* specifies the involved node and the time when the attack takes place.

The second section of the *ACF* consists of the list of conditional attacks to be reproduced. For each one of them, the following elements are specified: i) the time starting from which the attack occurs; ii) the set of nodes that perform the attack; iii) the packet filter; and, finally, iv) the list of *packet primitives* to be executed.

Finally, the third section of the *ACF* consists of the list of unconditional attacks to be reproduced. For each one of them, the following elements are specified: i) the time starting from which the attack occurs; ii) the occurrence frequency according to which the attack has to be performed over time; and, finally, iii) the list of *packet primitives* to be executed.

At its startup, the Attack Simulator retrieves the configurations of the attacks to be reproduced from the *ACF*. Then, the following data structures are initialized.

Physical attacks. The simulator considers every node n among the N nodes involved in at least one physical attack. For each node n, the simulator creates a list of physical attacks L_{PA}, containing N_{PA} elements. Such elements represent physical attacks that involve node n, and are ordered in a chronological fashion, according to their occurence time. Finally, the simulator starts N sets of timers, one for each node n. That is, for each node n, it starts N_{PA} timers, one for each attack in L_{PA}, thus scheduling their occurrence.

Conditional attacks. The simulator considers every node n among the N nodes involved in at least one conditional attack. For each node n, the simulator creates a list of conditional attacks L_{CA}, containing N_{CA} elements. Such elements represent conditional attacks performed by node n, and are ordered in a chronological fashion, according to their starting time. Each element includes: i) the packet filter; and ii) the list of events that define the attack. Finally, the simulator prepares N sets of timers, one for each of the above mentioned nodes. Then, for each node n, it starts N_{CA} timers, one for each attack in L_{CA}, thus scheduling the beginning of their occurrence.

Unconditional attacks. The simulator creates a list of unconditional attacks L_{UA}, containing N_{UA} elements. Such elements represent unconditional attacks to be reproduced, and are ordered in a chronological fashion, according to their starting time. Each one of them includes: i) the occurrence frequency according to which the attack is reproduced over time; and ii) the list of events that define the attack. Finally, the simulator starts N_{UA} timers, one for each attack in L_{UA}, thus scheduling their first occurrence.

Then, the Attack Simulator relies on the above mentioned data structures, and reproduces attacks as follows.

Physical attacks. When the timer associated to a physical attack expires, the simulator retrieves the corresponding attack A from the right L_{PA} list, and reproduces it on the involved node. Then, the attack A is removed from the L_{PA} list.

Conditional attacks. When the timer associated to a conditional attack expires, the simulator retrieves the corresponding attack A from the right L_{CA} list, associated to node n. From then on, node n starts to intercept packets by means of its own Local Filter, and filters them according to the packet filter specified in A. Then, for each

packet that satisfies the packet filter, node n executes the list of events defined in A. The actual reproduction of conditional attacks may involve the Global Filter, as well as more than one Local Filter module.

Unconditional attacks. When the timer associated to an unconditional attack expires, the simulator retrieves the corresponding attack A from the L_{UA} list. From then on, the list of events defined in A is executed, and repeatedly performed according to the specified occurrence frequency. The Global Filter is responsible for starting the actual reproduction of unconditional attacks.

It is worth remarking that we are interested in simulating and evaluating the *effects* of attacks. That is, we assume that attacks have been entirely and successfully performed, and focus on their effects on the WSN. Unlike described in [108], we do not simulate the actual occurrence of attacks. Instead, we reproduce their final effects on the sensor network.

Finally, it is worth noting that our architectural model is general, and can be further extended. In fact, the Local Filter module is able to deal with arbitrary compositions of communication layers. This makes ASF potentially suitable for any discrete event network simulator.

6.2.4 ASF prototype implementation

We implemented a prototype of ASF for the WSNs simulator Castalia [2], based on the OMNeT++ platform [3]. Castalia considers the network as a collection of nodes, which sense values according to a given physical process, and communicate through a commonly shared wireless channel.

In the original architecture of Castalia, nodes are composed of different submodules. Sensor nodes applications interact with the physical process through a sensor manager module, and retrieve physical information from the environment. Also, nodes are provided with a full communication stack, composed by a Routing, a MAC, and a Radio layer. Thanks to such communication modules, the application sends (receives) packets to (from) the wireless channel. Also, Castalia provides the implementations of different Routing and MAC layers.

Our ASF implementation for Castalia improves the original node architecture by introducing the Local Filter module. As shown in Figure 6.4, the Local Filter intercepts incoming and outgoing packets travelling through the communication stack, between every pair of layers. Also, every node's Local Filter module is connected with the Global Filter module.

6.3 Application scenario

We consider a room, whose size is 100 by 20 meters. The room is composed by three different areas, and each one of them hosts a heater, i.e. *S1*, *S2*, and *S3*. Every heater has a constant temperature of 200 °C.

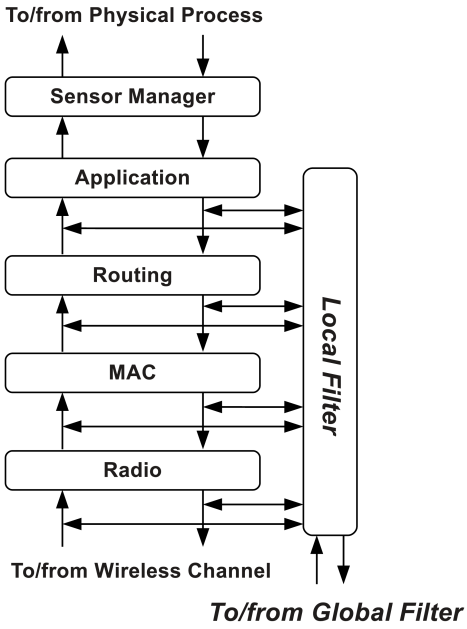


Figure 6.4. Enhanced architecture of Castalia node.

In order to monitor the heat level in a regular way, eighteen static sensor nodes $n_1, n_2, \ldots, n_{18}$ are positioned within the room in a regular fashion. That is, they form three rows of six nodes each. The network is divided into three different node *clusters*, each one covering a different area, i.e. $\{n_1, n_2, n_7, n_8, n_{13}, n_{14}\}$, $\{n_3, n_4, n_9, n_{10}, n_{15}, n_{16}\}$, and $\{n_5, n_6, n_{11}, n_{12}, n_{17}, n_{18}\}$. Basically, each node is responsible for periodically monitoring the temperature in its cluster.

An additional node is placed in the very center of the room, and acts as *sink*. The sink node collects reports received by other nodes, and computes average temperatures over time. In particular, it computes and stores average temperatures for each one of the three clusters, as well as for the whole room.

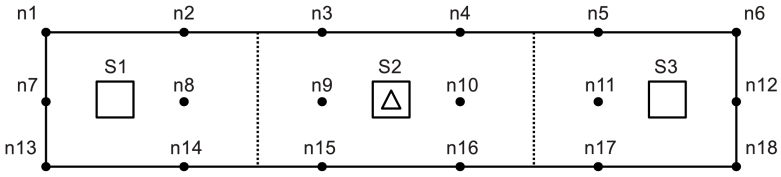


Figure 6.5. Application scenario and network topology.

Figure 6.5 depicts the room described above. The heaters are represented by squares, one for each area. The sink node is the triangle above the central heater. Finally, the eighteen black circles represent the sensor nodes deployed in the room.

6.3.1 Threat model

In such a scenario, an adversary may be interested in compromising the service availability, or altering reports correctness before they are collected by the sink. We consider four possible attacks to the monitoring wireless sensor network.

Removal attack. The adversary removes a number of sensor nodes from the network. By doing so, the computation of average temperatures is clearly altered. However, this attack is relatively simple to detect, since the sink is supposed to not receive temperature reports from removed nodes anymore.

Misplace attack. The adversary moves a number of sensor nodes from their original position to a new one. By properly choosing the new positions, it is possible to alter the computation of average temperatures on the sink node, or even worse. In fact, if a temperature warning threshold is defined, an alarm can be erroneously raised, because of a dangerously fake high temperature. This attack is far more difficult to detect, since the sink assumes that all nodes' original positions are unchanged.

Reprogram attack. The adversary tampers with a number of sensor nodes, and reprograms their behavior. In particular, she can force them to erroneously report either the minimum or the maximum possible temperature value, or even a randomly generated value. In the latter case, the random value is comprised among the minimum and maximum possible value. This attack is quite hard to be detected, although comparisons with other nodes' reports may help to contrast its effectiveness.

Drop attack. The adversary forces a number of sensor nodes to drop a subset of packets, according to specified criteria. For instance, a node can be forced to drop all data packets sent to the sink by a certain group of senders. This attack is quite hard to be detected, since packet loss may be erroneously considered due to packet corruption or medium access issues.

6.4 Simulative analysis

In this section, we refer to the application scenario described in Section 6.3, with network nodes sensing temperature in their proximity one time per second. Then, we evaluate the effects of the four attacks presented in Section 6.3.1 by using our prototype implementation of ASF for the Castalia simulator.

In our simulations, we consider the Multipath Rings routing protocol [1], the IEEE 802.15.4 MAC protocol [52], and the CC2420 radio chipset [99]. Results were obtained by means of 10 simulation runs, whose length was 600 seconds each.

The heat propagation model is based on the *Customizable Physical Process* provided by Castalia [1]. We assume that sensor nodes are able to report 0 ℃ and 1000 ℃ as minimum and maximum temperature value, respectively.

In the following, we consider the attacks described in Section 6.3.1. For each one of them, we provide an example of attack configuration using *ASL* primitives presented in Section 6.2.1, and discuss its impact on the application.

Considered attacks occur at time $t = 200$ s, and may involve three different pairs of nodes. In particular, the adversary manages to compromise either i) not a node; ii) nodes {2,3}; iii) nodes {9,10}; or iv) nodes {12,18}.

With reference to Figure 6.5, nodes 9 and 10 are supposed to be the most important ones, being very close to the central heater *S2* and the sink node. On the other hand, nodes 12 and 18 are supposed to be less important, being positioned at the extreme right side of the room and far from the sink node.

6.4.1 Removal attack

At time $t = 200$ s, the adversary removes either nodes {2,3}, {9,10}, or {12,18} from the network. In case nodes {2,3} are removed, the attack can be described as follows.

```
destroy(2,200);
destroy(3,200);
```

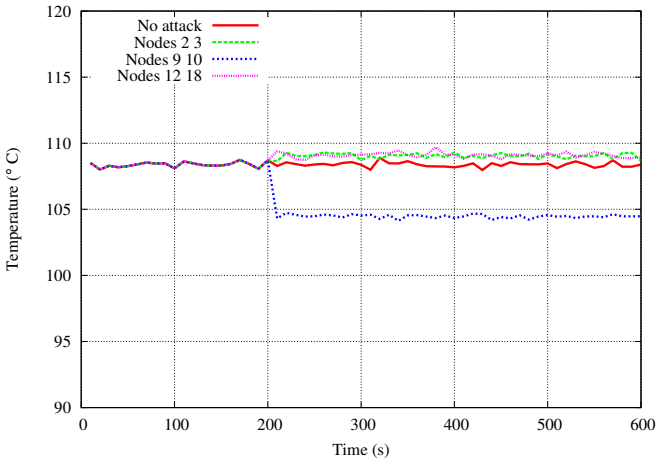


Figure 6.6. Average room temperature with node removal.

Figure 6.6 shows variations of the average perceived room temperature, in case different pairs of nodes are removed. It is evident that the average value remains almost the same, and barely changes only if nodes $\{9, 10\}$ are removed.

This result is reasonable, since the Multipath Rings routing protocol introduces redundancy in network connectivity, which results to be pretty robust. Then, messages sent by a given node are likely to be successfully delivered to the sink, through some of the other nodes acting as relay.

Actually affecting temperature monitoring requires to remove either a consistent amount of nodes, or the most relevant ones. Observable effects can be seen if nodes $\{9, 10\}$ are removed, because of their proximity to the sink node, and their major contribution in the packet relaying process.

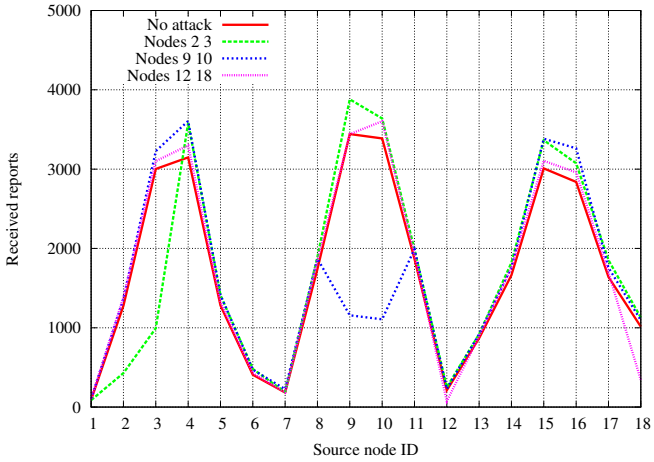


Figure 6.7. Reports reception with node removal.

Figure 6.7 shows the amount of temperature reports received by the sink from each node. The graph confirms that the amount of received reports can be drastically reduced, depending on the specific pair of nodes removed from the network. Also, if a pair of nodes is removed, reports from other nodes are more likely to be delivered to the sink. This is due to a reduced number of nodes contending to access the medium.

6.4.2 Misplace attack

At time $t = 200$ s, the adversary moves either nodes {2,3}, {9,10}, or {12,18} to the specified new position. More specifically: i) nodes {2,3} are moved to position (0;10;0); ii) nodes {9,10} are moved to position (100;10;0); and, finally, iii) nodes {12,18} are moved to position (50;10;0). In case nodes {2,3} are moved to position (0;10;0), the attack can be described as follows.

```
move(2,200,0,10,0);
move(3,200,0,10,0);
```

Figure 6.8 shows variations of the average perceived room temperature, in case different pairs of nodes are misplaced. As depicted in the graph, the nodes pair {12,18} is the only one that not negligibly affects the average temperature perceived in the room. This is because such two nodes are supposed to be at the room border and distant from the sink. Then, they are moved to the room center, at the very same position of the sink and the central heater *S2*, thus altering the average perceived temperature of up 20%.

If we focus on the right cluster, the misplace attack appears to be much more effective. As shown in Figure 6.9, nodes {12,18} are still the only ones that consistently affect the average temperature in the right cluster, if moved away from their legitimate

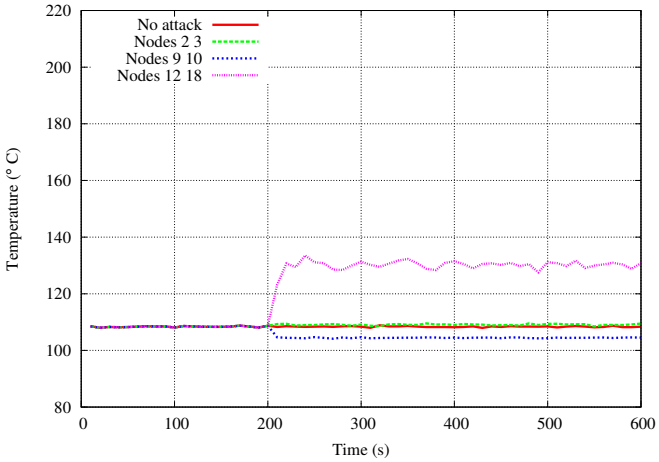


Figure 6.8. Average room temperature with node misplacing.

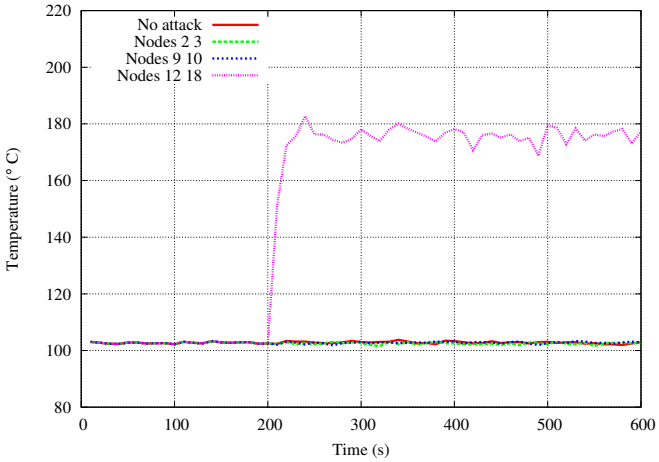


Figure 6.9. Average right cluster temperature with node misplacing.

positions. However, the average temperature in the right cluster is erroneously perceived as considerably different, i.e. up to 70% higher. Clearly, if a monitoring system took into account also single clusters, this attack might easily succeed in improperly raising an alarm.

6.4.3 Reprogram attack

At time $t = 200$ s, the adversary reprograms either nodes $\{9,10\}$ or $\{12,18\}$ to change the content of their own application data packets, i.e. their temperature reports, before sending them to the sink. The new content can be either the minimum, the maximum,

or a random temperature value. In case nodes {9,10} replace their reported temperature with a random value, the attack can be described as follows.

```
from t = 200; nodes = "9 10" do {
 if(packet.APP.source==SELF &&
    packet.APP.type==DATA)
      change(packet,APP.payload,RAND);
}
```

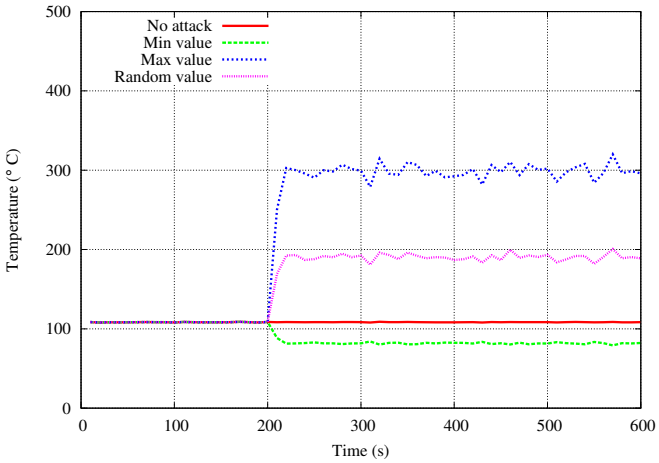


Figure 6.10. Average room temperature with nodes {9, 10} reprogrammed.

Figure 6.10 shows variations of the average perceived room temperature, in case nodes {9, 10} are reprogrammed. As depicted in the graph, forcing nodes to provide a random temperature value results in erroneously perceiving the room temperature up to 100% higher. Moreover, if the nodes report the maximum possible value, the average perceived room temperature is over 200% higher. Thus, by performing this attack, it is easy to improperly raise an alarm within the room.

As shown in Figure 6.11, this attack is less severe in case nodes {12, 18} are reprogrammed. In fact, such nodes are located at the right room border, i.e. far from the sink, and not so close to an heater (especially node 18), i.e. they are less influent in the temperature monitoring process. Thus, if nodes are forced to report the maximum possible value, the average perceived room temperature can be up to 40% higher.

6.4.4 Drop attack

At time $t = 200$ s, nodes {9,10} start to retain all MAC data packets sent by nodes {1,2,7,8,13,14}. Then, they discard such packets, rather than relaying them to the sink according to the Multipath Rings routing protocol. In case nodes {9,10} are forced to drop MAC data packets from other nodes, the attack can be described as follows. The

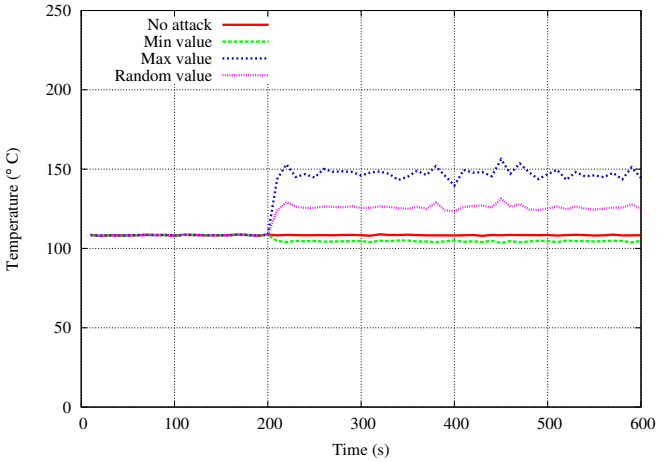


Figure 6.11. Average room temperature with nodes $\{12, 18\}$ reprogrammed.

utility function *belong()* returns *true* in case the list specified as first argument includes the value specified as second argument.

```
srcList={1,2,7,8,13,14};
from t = 200; nodes = "9 10" do {
 if(belong(srcList,packet.MAC.source) &&
    packet.MAC.type==DATA)
      drop(packet);
}
```

Even if nodes {9,10} drop MAC data packets, the amount of reports received by the sink is basically not affected. As a consequence, the average perceived temperature within the room remains the same, even if a *Drop attack* occurs. We performed more simulations, where either nodes {2,3} or {12,18} perform the attack, and obtained similar results.

This is mainly due to the adopted Multipath Rings routing protocol. As discussed for the *Removal attack* in Section 6.4.1, sensor nodes act as relay at the network layer, thus introducing redundancy in network connectivity. Thus, even if temperature reports from sensor nodes are dropped, they are likely to be eventually delivered to the sink. This means that, in the considered application scenario, two nodes dropping MAC data packets are not sufficient to alter network activity and application performance in a relevant way.

Also, we think that if non trivial routing mechanisms are adopted, an attack consisting only in discarding packets can hardly affect applications and performance. Instead, discarding packets may be effective as part of more complex attacks, together with other basic actions (e.g. altering packet contents, reprogramming nodes, injecting fake packets).

7

Conclusion

In this Ph.D. dissertation, we have considered, highlighted, and supported the importance of security in Wireless Sensor Networks (WSNs). Specifically, we have stressed that securing WSNs displays peculiar issues with respect to other kinds of networks. This is mainly due to the scarce resources provided by typical sensor devices, in terms of memory capacity, computational capabilities, and battery lifetime.

Therefore, it is vital that adoption of security solutions trades between their effectiveness and efficiency. That is, adopted countermeasures have to be scalable with the network size, and affordable by resource-scarce devices. Also, their impact on network performance must be sustainable in terms of communication overhead. In addition, adopted solutions should be able to deal with different traffic flows, and easily adapt to possible changes in network conditions. Finally, countermeasures should be chosen and configured according to severity of security attacks, whose impact and effects should be known since from the network design phase.

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.
- Analysis and solutions against deficiencies of ZigBee security services.
- A highly scalable scheme for group rekeying including collusion management.
- A transparent and reconfigurable software component for secure communication.
- A framework for quantitative evaluation of security attacks through simulation.

We believe that such contributions represent a valid set of recommendations, solutions, and tools to properly secure applications based on WSNs, and hope they will be a first step towards further achievements in this field.

References

1. Castalia - A simulator for Wireless Sensor Networks and Body Area Networks, Version 3.2 User's Manual. `http://castalia.npc.nicta.com.au/pdfs/Castalia-UserManual.pdf`.
2. National ICT Australia - Castalia. `http://castalia.npc.nicta.com.au/`.
3. OMNeT++ Network Simulation Framework. `http://www.omnetpp.org/`.
4. Source code of the IEEE 802.15.4 MAC Implementation. `http://code.google.com/p/tinyos-main/source/browse/trunk/tos/lib/mac/tkn154/`.
5. Source code of the IEEE 802.15.4 security sublayer implementation. `http://www.iet.unipi.it/g.dini/download/code/154_security_sublayer.zip`.
6. Texas Instruments CC2430DB.
7. A. A. Cardenas, T. Roosta and S. Sastry. Rethinking security properties, threat models, and the design space in sensor networks: A case study in SCADA systems. *Ad Hoc Networks*, 7(8):1434–1447, 2009.
8. A. Becher, E. Becher, Z. Benenson and M. Dornseif. Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks. In *Proceedings of the 3rd International Conference on Security in Pervasive Computing (SPC 2006)*, pages 104–118, 2006.
9. A. Biryukov. Skipjack. In H. C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, Berlin, Germany, 2005.
10. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, USA, October 2001.
11. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pages 245–256, St. Louis, Missouri, USA, April 2008.
12. A. Perrig, R. Szewczyk, V. Wen, D. Culler and D. J. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, pages 189–199, New York, NY, USA, July 2001. ACM.
13. A. T. Sherman and D. A. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, May 2003.
14. A. Wander, N. Gura, H. Eberle, V. Gupta and S. C. Shantz. Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 324–328, Washington, DC, USA, March 2005. IEEE Computer Society.
15. B. Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. Internet Engineering Task Force, Fremont, CA, USA, 2000.

16. B. Latré, P. De Mil, I. Moerman, N. Van Dierdonck, B. Dhoedt and P. Demeester. Maximum Throughput and Minimum Delay in IEEE 802.15.4. In *Proceedings of the First international conference on Mobile Ad-hoc and Sensor Networks*, volume 3794, pages 866–876, Berlin, Germany, December 2005. Springer.
17. B. Sun, K. Wu and U. Pooch. Secure Routing Against Black-hole Attack in Mobile Ad Hoc Networks. In *Proceedings of the International Conference on Communications and Computer Networks (CCN '02)*, November 2002.
18. B. Sun, L. Osborne, X. Yang and S. Guizani. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wireless Communications*, 14(5):56–63, 2007.
19. B. W. Lampson. Hints for Computer System Design. *ACM Operating Systems Review*, 17(5):33–48, October 1983.
20. C. Alcaraz, R. Roman and J. Lopez. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Network & Applications*, 12(4):231–244, August 2007.
21. C. H. Lim. LEAP++: A Robust Key Establishment Scheme for Wireless Sensor Networks. In *Proceedings of the Distributed Computing Systems Workshops*, pages 376–381, Washington, DC, USA, June 2008. IEEE Computer Society.
22. C. Hartung, R. Han, J. Deng and S. Mishra. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of the 1st IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks*, pages 289–299, Washington, DC, USA, September 2005. IEEE Computer Society.
23. C. K. Wong, M. Gouda and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, February 2000.
24. C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 113–127, May 2003.
25. C. Karlof, N. Sastry and D. Wagner. TinySec: a link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded Networked Sensor Systems*, SenSys '04, pages 162–175, 2004.
26. C. Pfleeger and S. L. Pfleegeer. *Security in Computing–4th Edition*. Prentice Hall, Upper Saddle River, NJ, USA, 2006.
27. C.E. Perkins and E.M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, pages 90–100, February 1999.
28. Certicom Research. *Standards for Efficient Cryptography: SEC 1 (working draft) ver 1.7: Elliptic Curve Cryptography*, November 2006.
29. Certicom Research. *Standards for Efficient Cryptography: SEC 4 (working draft) ver 1.1r1: Elliptic Curve Cryptography*, June 2006.
30. CONET. Cooperating Objects NETwork of excellence, European Commission, 7th Framework Programme, Grant Agreement n. 224053. `http://www.cooperating-objects.eu/`, 2008.
31. Crossbow Technology Inc. MPR/MIB User's Manual, August 2004.
32. D. Eastlake and P. Jones. RFC 3174. `http://tools.ietf.org/html/rfc3174`, September 2001.
33. D. Wallner, E. Harder and R. Agee. *Key Management for Multicast: Issues and Architectures*. Internet Engineering Task Force, Fremont, CA, USA, 1999.
34. Daintree Networks. *Understanding ZigBee Commissioning*, January 2007. `http://www.daintree.net`.
35. Daintree Networks. *Getting Started with ZigBee and IEEE 802.15.4*, February 2008. `http://www.daintree.net/`.
36. E. Cole. *Network Security Bible, 2nd Edition*. Wiley Publishing Inc., Indianapolis, IN, USA, September 2009.

37. F. Amini, M. Khan, J. Mišić and H. Pourreza. Performance of IEEE 802.15.4 Clusters with Power Management and Key Exchange. *Journal of Computer Science and Technology*, 23:377–388, 2008.
38. F. Bao, I. Chen, M. Chang and J. Cho. Hierarchical Trust Management for Wireless Sensor Networks and its Applications to Trust-Based Routing and Intrusion Detection. *IEEE Transactions on Network and Service Management*, 9(2):1–15, 2012.
39. G. Anastasi, M. Conti, M. Di Francesco and A. Passarella. Energy Conservation in Wireless Sensor Networks: A survey. *Ad Hoc Networks*, 7(3):537–568, May 2009.
40. G. Dini and I. M. Savino. An efficient key revocation protocol for wireless sensor networks. In *Proceedings of the 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 450–452, Washington, DC, USA, June 2006. IEEE Computer Society.
41. G. Dini and I. M. Savino. S2RP: a Secure and Scalable Rekeying Protocol for wireless sensor networks. In *Proceedings of the 2006 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2006)*, pages 457–466, October 2006.
42. G. Dini and I. M. Savino. S2RP: a Secure and Scalable Rekeying Protocol for Wireless Sensor Networks. In *Proceedings of the 2006 IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 457–466, Washington, DC, USA, October 2006. IEEE Computer Society.
43. G. Dini and I. M. Savino. A Security Architecture for Reconfigurable Networked Embedded Systems. *International Journal of Wireless Information Networks*, 17:11–25, 2010.
44. G. Dini and I. M. Savino. LARK: A Lightweight Authenticated ReKeying Scheme for Clustered Wireless Sensor Networks. *ACM Transactions on Embedded Computing Systems*, 10(4):41:1–41:35, November 2011.
45. G. Dini and M. Tiloca. Considerations on Security in ZigBee Networks. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC 2010)*, pages 58–65, June 2010.
46. G. Dini and M. Tiloca. ASF: An attack simulation framework for wireless sensor networks. In *Proceedings of the IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2012)*, pages 203–210, October 2012.
47. G. Dini and M. Tiloca. HISS: a HIghly Scalable Scheme for group rekeying. *The Computer Journal*, 56(4):508–525, April 2013.
48. G. Gaubatz, J.-P. Kaps, E. Öztürk and B. Sunar. State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 146–150, Washington, DC, USA, March 2005. IEEE Computer Society.
49. G. Padmavathi and D. Shanmugapriya. A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks. *International Journal of Computer Science and Information Security*, 4(1 & 2):117–125, August 2009.
50. I. M. Atakli, H. Hu, Y. Chen, W. S. Ku and Z. Su. Malicious node detection in wireless sensor networks using weighted trust evaluation. In *Proceedings of the 2008 Spring simulation multiconference*, SpringSim '08, pages 836–843, San Diego, CA, USA, 2008. Society for Computer Simulation International.
51. Institute of Electrical and Electronics Engineers, Inc., New York. *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)*, September 2003.
52. Institute of Electrical and Electronics Engineers, Inc., New York. *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)*, September 2006.

53. J. Büsch, R. Daidone, J.-H. Hauer, R. Severino, S. Tennina and M. Tiloca. Poster Abstract: An Open-Source IEEE 802.15.4 MAC Implementation for TinyOS 2.1. In *Poster and Demo Session of the 8th European Conference on Wireless Sensor Networks (EWSN 2011)*, pages 69–70, February 2011.
54. J. Doumen, Y. W. Law and P. H. Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks*, 2(1):65–93, February 2006.
55. J. Fan, P. Judge and M. H. Ammar. HySOR: group key management with collusion-scalability tradeoffs using a hybrid structuring of receivers. In *Proceedings of the Eleventh International Conference on Computer Communications and Networks*, pages 196–201, Washington, DC, USA, October 2002. IEEE Computer Society.
56. J. Ma, W. Wang and S. J. Moon. CRMS: A Collusion-Resistant Matrix System for Group Key Management in Wireless Networks. In *Proceedings of the 2008 IEEE International Conference on Communications*, pages 1551–1555, Washington, DC, USA, May 2008. IEEE Computer Society.
57. J. Maerien, S. Michiels, C. Huygens and W. Joosen. 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 (WiMob 2010)*, pages 121–128, October 2010.
58. J. Mišić. Cost of secure sensing in IEEE 802.15.4 networks. *IEEE Transactions on Wireless Communications*, 8(5):2494–2504, 2009.
59. J. Newsome, E. Shi, D. Song and A. Perrig. The Sybil attack in sensor networks: analysis & defenses. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, pages 259–268, April 2004.
60. J.-S. Lee. Performance evaluation of IEEE 802.15.4 for low-rate wireless personal area networks. *IEEE Transactions on Consumer Electronics*, 52(3):742–749, August 2006.
61. J. Zhang, M. Zulkernine and A. Haque. Random-Forests-Based Network Intrusion Detection Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(5):649–659, September 2008.
62. J. Zheng and M. J. Lee and M. Anshel. Toward Secure Low Rate Wireless Personal Area Networks. *IEEE Transactions on Mobile Computing*, 5(10):1361–1373, 2006.
63. K. Ghumman, M. Younis and M. Eltoweissy. Key management in wireless ad hoc networks: collusion analysis and prevention. In *Proceedings of the 24th IEEE International Conference on Performance, Computing, and Communications*, pages 199–203, Washington, DC, USA, April 2005. IEEE Computer Society.
64. L. Buttyan, D. Gessner, A. Hessler and P. Langendoerfer. Application of wireless sensor networks in critical infrastructure protection: challenges and design options. *IEEE Wireless Communications*, 17(5):44–49, October 2010.
65. L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.
66. M. Albano, S. Chessa and R. Di Pietro. Information Assurance in Critical Infrastructures via Wireless Sensor Networks. In *Proceedings of the Fourth International Conference on Information Assurance and Security (ISIAS '08)*, pages 305–310, 2008.
67. M. Chorzempa, J.-M. Park and M. Eltoweissy. SECK: survivable and efficient clustered keying for wireless sensor networks. In *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference*, pages 453–458, Washington, DC, USA, April 2005. IEEE Computer Society.
68. M. Chorzempa, J.-M. Park and M. Eltoweissy. Key management for long-lived sensor networks in hostile environments. *Computer Communications*, 30(9):1964–1979, 2007.
69. M. Eltoweissy, A. Wadaa, S. Olariu and L. Wilson. Group key management scheme for large-scale sensor networks. *Ad Hoc Networks*, 3(5):668–688, 2005.

70. M. Eltoweissy, M. H. Heydari, L. Morales and I. H. Sudborough. Combinatorial Optimization of Group Key Management. *Journal of Network and Systems Management*, 12(1):33–50, March 2004.
71. M. F. Younis, K. Ghumman and M. Eltoweissy. Location-Aware Combinatorial Key Management Scheme for Clustered Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):865–882, August 2006.
72. M. T. Goodrich, J. Z. Sun and R. Tamassia. Efficient Tree-Based Revocation in Groups of Low-State Devices. In *Proceedings of CRYPTO '04, Volume 2204 of LNCS*, pages 511–527, Berlin, Germany, August 2004. Springer.
73. M. Waldvogel, G. Caronni, D. Sun, N. Weiler and B. Plattner. The VersaKey Framework: Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, September 1999.
74. Moteiv Corporation. *Tmote iv Low Power Wireless Sensor Module*. Moteiv Corporation, San Francisco, CA, USA, November 2006. `http://www.cs.jhu.edu/~cliang4/public/datasheets/tmote-sky-datasheet.pdf`.
75. N. Matthys, C. Huygens, D. Hughes, S. Michiels and W. Joosen. 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 (POLICY 2012)*, pages 53–60, July 2012.
76. N. Sastry and D. Wagner. Security considerations for IEEE 802.15.4 networks. In *Proceedings of the 3rd ACM workshop on Wireless security*, WiSe '04, pages 32–42, New York, NY, USA, 2004. ACM.
77. Naor, D., Naor, M. and Lotspiech, J. B. Revocation and Tracing Schemes for Stateless Receivers. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 41–62, Berlin, Germany, August 2001. Springer.
78. National Institute of Standards and Technology. *Federal Information Processing Standards Publication 197, Specification for the ADVANCED ENCRYPTION STANDARD (AES)*, November 2001.
79. National Institute of Standards and Technology (NIST). *Secure Hash Standard*. National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2008. `http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf`.
80. P. E. Lanigan, R. Gandhi and P. Narasimhan. Sluice: Secure Dissemination of Code Updates in Sensor Networks. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006)*, pages 53–62, July 2006.
81. P. Tague and R. Poovendran. Modeling node capture attacks in wireless sensor networks. In *Proceedings of the 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 1221–1224, September 2008.
82. 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.
83. R. A. Kemmerer and G. Vigna. Hi-DRA: Intrusion Detection for Internet Security. *Proceedings of the IEEE*, 93(10):1848–1857, October 2005.
84. R. Daidone. Experimental Evaluations of Security Impact on IEEE 802.15.4 Networks. In *PhD Forum at IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2011)*, June 2011.
85. R. Daidone, G. Dini and M. Tiloca. 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 2011)*, pages 1–6, June 2011.
86. R. Daidone, G. Dini and M. Tiloca. STaR implementation source code. `http://www.iet.unipi.it/g.dini/download/code/star.zip`, 2012.

87. R. Daidone, G. Dini and M. Tiloca. STaR: Security Transparency and Reconfigurability for Wireless Sensor Networks programming. In *Proceedings of the 2nd International Conference on Sensor Networks (SENSORNETS 2013)*, pages 1–6, February 2013. To appear.
88. R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd Edition*. Wiley Publishing Inc., Indianapolis, IN, USA, April 2008.
89. S. Han, E. Chang, L. Gao and T. Dillon. Taxonomy of Attacks on Wireless Sensor Networks. In Blyth, Andrew, editor, *EC2ND 2005*, pages 97–105. Springer London, 2006.
90. S. Hyun, P. Ning, A. Liu and W. Du. Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks. In *Proceedings of the 2008 International Conference on Information Processing in Sensor Networks (IPSN '08)*, pages 445–456, April 2008.
91. S. Kaplantzis, A. Shilton, N. Mani and Y. A. Sekercioglu. Detecting Selective Forwarding Attacks in Wireless Sensor Networks using Support Vector Machines. In *Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information (ISSNIP 2007)*, pages 335–340, December 2007.
92. S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys*, 35(3):309–329, September 2003.
93. S. Rajasegarar, C. Leckie and M. Palaniswami. Anomaly detection in wireless sensor networks. *IEEE Wireless Communications*, 15(4):34–40, 2008.
94. S. Setia, S. Zhu and S. Jajodia. LEAP+: Efficient security mechanisms for large-scale distributed sensor networks. *ACM Transactions on Sensor Networks*, 2(4):500–528, November 2006.
95. S. Tennina, A. Koubaa, R. Daidone, M. Alves, P. Jurcik, R. Severino, M. Tiloca, J.-H. Hauer, N. Pereira, G. Dini, M. Bouroche and E. Tovar. *IEEE 802.15.4 and ZigBee as enabling technologies for low-power wireless systems with Quality-of-Service constraints*. Lecture Notes in Electrical Engineering. Springer, Berlin, 2012. (To be published).
96. S. Xu. On the security of group communication schemes. *Journal of Computer Security*, 15:129–169, January 2007.
97. T. Bonaci, L. Bushnell and R. Poovendran. Node capture attacks in wireless sensor networks: A system theoretic approach. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC 2010)*, pages 6765–6772, December 2010.
98. T.-G. Lupu. Main types of attacks in wireless sensor networks. In *Proceedings of the 9th WSEAS international conference on signal, speech and image processing, and 9th WSEAS international conference on Multimedia, internet & video technologies*, SSIP '09/MIV'09, pages 180–185. World Scientific and Engineering Academy and Society (WSEAS), 2009.
99. Texas Instruments. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee ready RF Transceiver. `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`, 2012.
100. TinyOS Working Group. TinyOS Home Page. `http://www.tinyos.net/`, 2012.
101. U.S. National Security Agency (NSA). SKIPJACK and KEA algorithm specifications. `http://csrc.nist.gov/groups/ST/toolkit/documents/skipjack/skipjack.pdf`, May 1998.
102. V. B. Mišić, J. Fang and J. Mišić. MAC layer security of 802.15.4-compliant networks. *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*, pages 8 pp. –854, 2005.
103. W. Du, L. Fang and P. Ning. LAD: Localization Anomaly Detection forWireless Sensor Networks. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, April 2005.
104. W. Gu, N. Dutta, S. Chellappan and B. Xiaole. Providing End-to-End Secure Communications in Wireless Sensor Networks. *IEEE Transactions on Network and Service Management*, 8(3):205–218, September 2011.

105. W. R. Cheswick, S. M. Bellovin and A. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker (2nd Edition)*. Addison-Wesley Professional, Boston, MA, USA, 2003.
106. Y. Hu, A. Perrig and D. B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM 2003)*, pages 1976–1986, April 2003.
107. Y-T. Wang and R. Bagrodia. Scalable emulation of TinyOS applications in heterogeneous network scenarios. In *Proceddings of the IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS 2009)*, pages 140–149, October 2009.
108. Y.-T. Wang and R. Bagrodia. SenSec: A Scalable and Accurate Framework for Wireless Sensor Network Security Evaluation. In *Proceedings of the 31st International Conference on Distributed Computing Systems Workshops (ICDCSW 2011)*, pages 230–239, June 2011.
109. Y. Wang, X. Wang, B. Xie, D. Wang and D.P. Agrawal. Intrusion Detection in Homogeneous and Heterogeneous Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, 7(6):698–711, 2008.
110. Y. Xiao, H. H. Chen, B. Sun, R. Wang and S. Sethi. MAC security and security overhead analysis in the IEEE 802.15.4 wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2006:81–81, April 2006.
111. Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu and M. Galloway. A survey of key management schemes in wireless sensor networks. *Computer Communications*, 30(11-12):2314–2341, 2007.
112. Y. Xu, G. Chen, J. Ford and F. Makedon. Detecting Wormhole Attacks in Wireless Sensor Networks. In *Critical Infrastructure Protection, Post-Proceedings of the First Annual IFIP Working Group 11.10 International Conference on Critical Infrastructure Protection*, volume 253 of *IFIP*, pages 267–279. Springer, 2007.
113. Z. Su, C. Lin, F. Ren, Y. Jiang and X. Chu. An Efficient Scheme for Secure Communication in Large-Scale Wireless Sensor Networks. In *Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing*, volume 3, pages 333–337, January 2009.
114. ZigBee Alliance. *ZigBee Document 064309r04, Commissioning Framework*.
115. ZigBee Alliance. *ZigBee Home Automation Public Application Profile*, October 2007. `http://www.zigbee.org/`.
116. ZigBee Alliance. *ZigBee Document 053474r17, ZigBee Specification*, January 2008. `http://www.zigbee.org/`.
117. ZigBee Alliance. *ZigBee Smart Energy Profile Specification*, December 2008. `http://www.zigbee.org/`.

Acknowledgment

For my Ph.D. dissertation, I would like to thank:

My advisor, Prof. Gianluca Dini:
for having put me in front of huge responsibilities and duties;
for having trained, trusted, and supported me during these years;
for having taught me to approach, face, manage, lead, and enjoy research activies.

My mum and dad:
for having effectively been by my side, in spite of the distance;
for having supported me both as young researcher and as man;
for having kept admiration and trust in my skills and will, often more than I did.

The other three "Big Four" guys whom I am (and will always be) connected with, for having always been by my side, each one in his own way. You know who you are.

All friends with whom I have shared something during these years, for their interest and support: Augusto Maggio, Antonella Mangiapane, Antonio Marino, Sergio Marino, Fabio Uscidda, Virginia Riccucci, Alessandro "Zio" Pischedda, Alberto Secondi, Giovanna Marceddu, Francesco "Frank" Fruci, Caterina Catalano, Gian Mario Fruci, Alessio Di Giacomo, Matteo Di Giacomo, Giovanni Matteoli, Matteo "Mars" Peru, Valentina Meloni, Paolo Madeddu, Marcello Ditta, Carmine "Nino" Benedetto, Luca "Fittizio" Laudadio, Nicolas Polidori, Adriano Panipucci, Annalisa Porcu, Camilla Tanca, Martina Ulivi, Alessio Ruiu, Antonella Rizzo, Fabrizio Uras, Leonardo Loparco, Francesco Filia, Francesco Mattiello, Gabriele "Zanne" Zannerini, Giacomo Bolognesi, Francesco Magno, Carlo Sciacca, Alessio Di Sandro, Valentino "fun()" Stopponi, Mercedes Chan, Luigi Trevisi, Fabio Fosso, Franco Fois, Roberto Pinna, Marco Geymet, Katia Minioto, Isabella Mastino, Simona Franchi, Arianna Pinna, Nicolò Manca, Pietro Ganadu, Alessandra Carta, Marco "Stich" Puggioni, Giampiero "L'ungherese" Manca, Giampiero Meloni, Marco "Lord Goblin" Piu, Andrea Giribaldi, Davide Venerdini.

My colleagues, for all things I learnt from them, and the good time spent together: Stefano Abbate, Giuseppe Anastasi, Hakjeon Bang, Simone Brienza, Stefano Campanelli, Luca Cassano, Sena Efsun Cebeci, Daniel Cesarini, Mario Cimino, Eleonora D'Andrea, Roberta Daidone, Domenico De Guglielmo, Davide Di Baccio, Adriano Faggiani, Adriano Fagiolini, Ilaria Giannetti, Francesco Giurlanda, Alessandro Improta, Koteswararao "Koti" Kondepu, Mariantonietta Noemi La Polla, Valerio Luconi, Simone Martini, Daniele Migliorini, Chiara Orsini, Pericle Perazzo, Andrea Saracino, Giovanni Stea, Giacomo Tanganelli, Carlo Vallati, Antonio Virdis.

People I have worked with in the remit of the European Network of Excellence CONET and the European Integrated Project PLANET: Prof. Pedro José Marrón, Dr. Chia Yen Shih, Richard Figura, Songwei Fu, Prof. Vlado Handziski, Jan-Hinrich Hauer, Prof. Mário Alves, Dr. Stefano Tennina, Ricardo Severino, Prof. Thiemo Voigt, Prof. Luca Mottola, Shahid Raza, Prof. Aníbal Ollero, Prof. José Ramiro Martínez de Dios, Alberto De San Bernabé Clemente, Prof. D.K. Arvind, Dr. Janek Mann, Paolo Proietti, Claudio Malavenda, Fabio Luigi Bellifemine, Claudio Borean.

All people from the Swedish Institute of Computer Science (SICS), for their great welcome during my visit in November 2012: Prof. Thiemo Voigt, Shahid Raza, Dr. Simon Duquennoy, Dr. Prasant Kumar Misra, Niclas Finne, Zhitao He, Joel Höglund.

Giovanni Stea:
for having made me know and enjoy so many great things, including Q-Zar, mountain excursions, progressive rock bands, British shows, comedians, and must-read books; for having established and maintained with me a so vivid and pleasant cultural interchange on a wide range of topics.

The following bands and artists, for having been my Ph.D. soundtrack, sometimes from a live stage, sometimes from speakers, often just in my own head: AC/DC, Anthrax, Banco del Mutuo Soccorso, Black Label Society, Black Sabbath, Blind Guardian, Charlie Parker, Children of Bodom, Circus Maximus, David Lee Roth, Deep Purple, Dizzy Gillespie, Dream Theater, Echolyn, Elio e le Storie Tese, Elvenking, Eric Johnson, Extreme, Fates Warning, Flying Colors, Frédéric Chopin, Genesis, Guns N' Roses, Iron Maiden, Jaco Pastorius, Jason Becker, Jethro Tull, Jimi Hendrix, Joe Satriani, Joey Tafolla, John Coltrane, John Petrucci, Keith Jarrett, L.A. Guns, Led Zeppelin, Liquid Tension Experiment, Megadeth, Meshuggah, Metallica, Miles Davis, Motörhead, Mr. Big, Negacy, Nevermore, Opeth, Ornette Coleman, Ozzy Osbourne, Pain of Salvation, Pantera, Pink Floyd, Porcupine Tree, Premiata Forneria Marconi, Queensrÿche, Racer X, Rush, Slash, Slayer, Sonny Rollins, Spastic Ink, Steve Vai, Stratovarius, Symphony X, Tommy Emmanuel, Trivium, Vanden Plas, Vinnie Moore, Vision Divine, Watchtower, Whitesnake, Wolfgang A. Mozart, Yngwie J. Malmsteen.

The "Borderline" Club in Pisa, for the unforgettable nights and the people who gave life to great shows: Paul Gilbert, Billy Sheehan, Vinnie Moore, Ciro Manna, Andy Timmons, Eric Martin, Allan Holdsworth, Richie Kotzen, Pino Scotto, John Corabi, Elvenking, Lost Angels, J27, Il Tempio delle Clessidre, Bed Memories, Surfer Joe & his Boss Combo, Surrender the coast, Enkymosis, Versions, Egoband, Noah, Fall of Darkness, Kalidia, Free-Go, Wind Rose, Soundforce, Sikitikis, Nitro Junkies, Run Over, Eldritch, Diluve, Ophyra, Subhuman, Silver Bullets, Angustiati, Alcolica, The Machine, Division Bell, Prowlers, Bastards, Aberdeen Revenge, The Rose, The Osbourne, True Fighters, NoOne Band, Strange Illusion, Purple Night, Heartbreakers, Pearl Pusher.

The "Ex-Wide" Club in Pisa, for the great jazz talks, concerts, and jam sessions.

Gabriele Cocco and the photography group "Arte e Luce", for having woken up my eyes and made me enjoy such a beautiful and pleasant sensory dimension.

All must-visit places to eat/drink in Pisa: Orzo bruno, La bottega del gelato, Gelateria De Coltelli, Ir tegame, Euro Kebab, Wok World, Le scuderie, DAB, Il mediceo, La casa della panna, Al-Madina, La piadona, Il crudo, Lo spaventapasseri, Sottobosco.

The Eyjafjöll icelander volcano and its eruption in April 2010, for having made me stuck in Stockholm city, definitely one of the best place I have ever been in this world.

Adrian Wallwork, for his course "English for writing and presenting scientific papers".

Richard Dawkins, **Jared Diamonds**, and **Brian Greene**, for having shed so many additional light on three topics I care about so much.

Bill Hicks and **Doug Stanhope**, for their fun, smart, and unequaled comic shows.

George R. R. Martin, for his beautiful tales from the Seven Kingdoms.

Italian Subs Addicted and **www.spinoza.it** for the great time they gave me.

Paolo "eriadan" Aldighieri, for his amazing and brilliant comic strips.

Antonio Pitzoi, for having taught me two vital things: i) women are the greatest mystery on Earth; ii) whatever happens in your life, do not quit playing the guitar.

The Ibanez company, for having given birth to my three beloved electric guitars, namely Jenny, Wendy, and Violet.