UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XXV Ciclo

Change to survive: a Moving Target Defense approach to secure resource-constrained distributed devices

---

Author:
Alessandra De Benedictis

Supervisor:
Prof. Valentina Casola
Coordinator:
Prof. Franco Garofalo

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Department of Electrical Engineering and Information Technology

April 2013

"I hear and I forget. I see and I believe. I do and I understand."

Confucius

UNIVERSITY OF NAPLES FEDERICO II

Abstract

Faculty of Computer Engineering
Department of Electrical Engineering and Information Technology

Doctor of Philosophy

**Change to survive:
a Moving Target Defense approach to secure resource-constrained distributed devices**

by Alessandra De Benedictis

This doctoral thesis has been developed with the aim of defining a design methodology for monitoring architectures composed of resource-constrained devices (sensor nodes, FPGAs, smartphones...), able to take into account both functional and non-functional requirements. Even if our primary focus was on security, our activity was aimed at identifying a holistic approach able to meet even other *quality* requirements, such as performance and energy consumption, as they are fundamental in real world applications.

Security, performance and energy consumption requirements are closely related to one another and are often conflicting, and typically in complex real-world scenarios they change over time, thus requiring the ability to adapt dynamically. These features make the definition of a comprehensive approach very challenging in constrained networks, and require the introduction of a more flexible strategy to achieve security while preserving the overall quality of the system.

In order to cope with these issues, we proposed a reconfiguration approach based on the Moving Target Defense paradigm, an emergent technique aimed at continuously changing a system's attack surface for thwarting attacks. Such mechanisms increase the uncertainty, complexity, and cost for attackers, limit the exposure of vulnerabilities, and ultimately increase overall resiliency, with the result of decreasing the attack probability.

We defined a reconfiguration model for a generic embedded node, identifying some of the possible reconfigurable parameters – namely the firmware, the APIs and the cryptosystem adopted to secure exchanged data – and characterized a reconfiguration strategy, aimed at choosing the new configuration to activate based on given requirements. In

order to do that, we introduced a coverage-based security metric to quantitatively measure the level of security provided by each system configuration; such metric, along with the commonly adopted performance metrics, is used by the reconfiguration strategy to identify the configuration to activate in the system that best meets the current requirements.

In order to show the feasibility of our approach in real applications, we considered a Wireless Sensor Networks (WSNs) case study. We defined a reconfiguration model characterized by two different cryptosystems, based on Elliptic Curve Cryptography (ECC), at the security layer, and two different firmware versions at the physical layer. We developed and implemented two ad-hoc reconfiguration mechanisms to perform security-level and physical-level reconfiguration, and conducted specific analyses on the security layer to show how reconfiguration can help increase, or at least control, the security level provided by a system.

At this aim, we first analyzed the performance, consumption and intrinsic security level provided by the two considered cryptosystems, and then conducted theoretical and experimental evaluations to show that reconfiguration is effective in increasing the complexity for the attacker.

Current MTD designs lack quantitative metrics to measure the effectiveness of the proposed mechanisms in terms of enhanced security. We adopted the attack probability to indirectly measure the level of security provided by each configuration and show that our approach is capable of reducing the probability of successful attacks, compared to a baseline scenario where configurations are static.

Keywords: Moving Target Defense, Wireless Sensor Networks, Reconfiguration, Security Metrics.

Preface

Some of the research and results described in this Ph.D. thesis has undergone peer review and has been published in, or at the date of this printing is being considered for publication in, academic journals, books, and conferences. In the following I list all the papers developed during my research work as Ph.D. student.

1. A. De Benedictis, A. Gaglione, N. Mazzocca. *Securing a Tiered Re-Taskable Sensing System.* In Proceedings of the 6th International Conference on Information Assurance and Security (IAS 2010), August 23-25, 2010, Atlanta, Georgia, USA. IEEE Computer Society. pp. 260-264.2010.

2. A. De Benedictis, A. Gaglione, N. Mazzocca. *A Secure Architecture for Re-Taskable Sensing Systems.* In Journal of Information Assurance and Security, Vol. 6, Issue 4, 2011 pp. 240-247.

3. V. Casola, A. De Benedictis, A. Mazzeo, N. Mazzocca. *SeNsIM-SEC: security in heterogeneous sensor networks.* In Proceedings of the 6th Conference on Network Architectures and Information System Security (SAR-SSI 2011), May 17-21 2011, La Rochelle, France. IEEE Computer Society, pp.17-24.

4. V. Casola, A. De Benedictis, A. Drago, and N. Mazzocca. *Analysis and comparison of security protocols in wireless sensor networks.* In Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on, pages 52-56, oct. 2011.

5. V. Casola, A. De Benedictis, A. Drago, M. Esposito, F. Flammini and N. Mazzocca. *Securing freight trains for hazardous material transportation: a WSN-based monitoring system.* In International Defence and Homeland Security Simulation Workshop (DHSS 2012), in cooperation with the I3M 2012 MultiConference, September 19-21 2012, Wien, Austria.

6. M. Albanese, A. De Benedictis, S. Jajodia, and P. Shakarian. *A Probabilistic Framework for Localization of Attackers in MANETs.* In Proceedings of the 17th European Symposium on Research in Computer Security (ESORICS 2012), September 10-12 2012, Pisa, Italy.

7. M. Albanese, A. De Benedictis, S. Jajodia, and K. Sun. *A Moving Target Defense Mechanism for MANETs Based on Identity Virtualization.* Submitted to the 1st IEEE Conference on Communications and Network Security (CNS 2013)

8. M. Albanese, A. De Benedictis, S. Jajodia, D. Torrieri. *A Probabilistic Framework for Jammer Identification in MANETs.* Submitted to Ad Hoc Networks, Elsevier, January 2013

9. V. Casola, A. De Benedictis, A. Drago, and N. Mazzocca. *SeNsiM-SEC: secure sensor networks integration to monitor rail freight transport.* Submitted to the Special Issue *Situation Awareness: Theory and Methodology* - International Journal on System of Systems Engineering (Inderscience)

Contents

List of Figures

List of Tables

To all who believe in me

Chapter 1

Introduction

The research activity conducted during my Ph.D. program had the main goal of defining a design methodology for monitoring architectures composed of resource-constrained devices (sensor nodes, FPGAs, smartphones...), able to take into account both functional and non-functional requirements. Even if our primary focus was on security, our activity was aimed at identifying a holistic approach able to meet even other *quality* requirements, such as performance and energy consumption, as they are fundamental in real world applications.

The methodological approach adopted to achieve this goal encompasses two aspects: (i) introduction of a mechanism to quantitatively evaluate the overall *quality* of a system, defined in terms of security, performance and energy consumption, and (ii) implementation of proper strategies aimed at controlling the quality of a system during its operation.

Security, performance and energy consumption requirements are closely related to one another and are often conflicting: security protocols for example have an impact on the number and the size of exchanged messages and on the elaboration time, computational load and packet flow negatively impact on nodes lifetime and performance and, in turn, nodes lifetime affects failure rate. The design activity typically aims at fulfilling a subset of these requirements, sacrificing the others, to get feasible solutions in real applications: often, due to the HW/SW limitations of the considered devices, security must be sacrificed to performance.

Moreover, in complex real-world scenarios, security and performance requirements may change over time, thus requiring the ability to adapt dynamically: a specific solution adopted at deployment time may result no more adequate later, due for example to uncovered malicious activities detection, or to performance decay caused by battery exhaustion.

Finally, it must be considered that *security degrades over time*: each security solution is designed to cope with a specific set of attacks and provides an *intrinsic level of security*, depending on the cryptographic scheme, the algorithm, the length of the keys, etc.. After deployment, the probability of having a successful attack increases, as the system's attack surface, defined as "the subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack" [1], is exposed to attackers. The longer a systems attack surface is exposed to attackers, the greater is the attackers' opportunity to gain knowledge about the system, its topology and vulnerabilities, to complete an attack.

In order to be able to cope with the above discussed issues, a more flexible way to achieve security, inspired by the Moving Target Defense (MTD) paradigm [2–4], can be devised. MTD is based on continuously changing a system's attack surface in order to increase the attacker's uncertainty about it, and limit the exposure of vulnerabilities and opportunity for attack. An intuitive means to implement MTD is *reconfiguration*, that could be performed both at a physical level, that is by actually changing some system parameters (firmware, implemented cryptosystem, message format, etc.), and at a virtual level, that is by adopting mechanisms to give the attackers a virtual view of the system that does not correspond to the real one.

We introduced a MTD approach based on fine-grained physical reconfiguration, that takes into account the specific security and performance requirements depending on the deployment scenario while having in mind the HW and SW constraints of the nodes. We defined a reconfiguration model, identifying the reconfigurable parameters of a system, and characterized a reconfiguration strategy, aimed at

choosing the new configuration to activate, given input requirements. Finally, we devised different reconfiguration mechanisms for a WSN case study to practically execute the reconfiguration operation.

As the main goal of reconfiguration is to increase the overall *quality* of a system, proper metrics must be defined to measure the level of security provided by each configuration, as well as the overall performance and the energy consumption profile, in order to be able to compare different configurations and choose the most appropriate one to meet the given requirements. While several metrics exist and are commonly adopted to measure a system's performance and power consumption, providing a quantitative measure of the level of security of a system is not straightforward: some security metrics have been proposed in the literature, mostly based on the analysis of attack graphs or on risk quantification, but they are generally hard to link to the adopted definition of configuration, that is centered on the mechanisms that are available to enforce a subset of security requirements.

We adopted a different metric, based on the coverage of a set of known attacks: once the admissible configurations and the attacks of interest have been identified, it is possible to build an *Attack Coverage Table*, that helps define the level of security provided by each configuration, by identifying the degree of coverage of each configuration with respect to specific attacks, or equivalently with respect to specific requirements. As said, security metric should be combined with other metrics dealing with energy consumption, response time or memory occupancy, to give an overall score to each configuration. The resulting score can be used by the reconfiguration scheduler to choose the new configuration to implement in order to meet the given requirements.

Even though all networks composed of limited computational, storage and power resources share the same issues, the analysis we conducted has been focused on Wireless Sensor Networks (WSNs), that are widely adopted in critical scenarios and present several security issues to be addressed by means of proper security policies and mechanisms.

Different security mechanisms have been implemented to secure WSNs, primarily based on symmetric key cryptography and Elliptic Curve Cryptography (ECC)[5], designed in order to limit power consumption and the computational/storage effort. We analyzed some of the most recent solutions proposed in the literature to protect data exchanged among sensor nodes, and selected two available libraries, namely WMECC [6] and TinyPairing[7], both based on ECC, providing different levels of security and having different resource utilization profiles.

We adopted the two libraries to build a cryptosystem used to secure a monitoring application and conducted specific analyses aimed at:

- measuring and comparing performance and power consumption of the two different cryptosystems,
- defining the intrinsic level of security provided by each cryptosystem, based on the security requirements they are able to ensure (static security analysis),
- showing how reconfiguration can help decrease the probability of attack for a brute force attack case.

We designed two reconfiguration mechanisms for WSN, respectively able to reconfigure the cryptosystem adopted to secure the communication, and the firmware installed on sensor nodes. The theoretical and experimental analyses that we conducted showed that, by periodically reconfiguring the nodes, the proposed mechanisms are effective in increasing the complexity for the attacker and, consequently, in decreasing the probability of completing a successful attack when the reconfiguration frequency is properly chosen.

Although many interesting activities have been conducted with respect to the above discussed topics, several issues are still open and need to be investigated. Our future plans include the design of a fully-automated reconfiguration strategy capable of identifying the system configuration that can best meet specific, dynamically changing requirements in terms of security, performance and power consumption. In order to do this, an innovative security metric for the comparison of different configurations must be defined; the most challenging aspect is the

identification and modeling of the dependency relations existing between security and time, that constitutes a relevant and still unexplored topic. We also plan to perform a deep evaluation of the optimal reconfiguration frequency and to introduce automatic mechanisms to map the existing requirements onto the available configurations (technological mapping).

This thesis work is organized as follows.

Chapter 2 describes the MTD approach adopted to control the level of security of a monitoring architecture composed of resource-constrained devices. We define a reconfiguration model for embedded systems, characterize a reconfiguration strategy and discuss about mechanisms to enforce the new configuration. In this chapter we also discuss about the dependency of security on time, and show how reconfiguration can help increase, or at least control, not only the security level provided by a system, but also its performance and consumption. In order to measure the level of security provided by each configuration, we introduce a coverage-based security metric, relying upon an Attack Coverage Matrix, that identifies the attacks covered by each available configuration.

Chapter 3 addresses the main security issues arising in networks composed of resource-constrained devices, focusing on the WSN case study. An overview of WSNs is given: the peculiar HW/SW features of sensor nodes are described, highlighting the main consequent challenges for security and performance, and the most common attacks against WSNs are presented, along with some of the most relevant security solutions proposed in the literature.

Chapter 4 describes the experimental setup: an overview is given of the WM-ECC and TinyPairing libraries, selected among the most recent solutions based on ECC to build a cryptosystem on top of a reference monitoring application. The chapter presents and discusses the analysis conducted on performance, consumption and intrinsic security of these two cryptosystems.

Chapter 5 shows the theoretical and experimental analysis conducted to prove that the proposed reconfiguration mechanisms are effective in increasing the complexity

for the attacker and, consequently, in decreasing the probability of completing a successful attack.

Finally, some concluding remarks and future directions are given in Chapter 6.

Chapter 2

An MTD approach to secure resource-constrained distributed devices

In this chapter, we describe the MTD approach adopted to control the level of security of a monitoring architecture composed of resource-constrained devices. We define a reconfiguration model for embedded systems, characterize a reconfiguration strategy and discuss about mechanisms to enforce the new configuration. We discuss about the dependency of security on time, and show how reconfiguration can help increase, or at least control, not only the security level provided by a system, but also its performance and consumption. We also introduce a coverage-based security metric, relying upon an Attack Coverage Matrix, that identifies the attacks covered by each available configuration.

2.1 Moving Target Defense

In recent years, we have witnessed a growing interest in techniques aimed at continuously changing a system's attack surface in order to prevent or thwart attacks. This approach to cyber defense is generally referred to as Moving Target Defense (MTD)[3, 4], and it is currently considered one of the *game-changing* themes in

cyber security by the Executive Office of the President, National Science and Technology Council. As stated in [2], Moving Target Defense "enables us to create, analyze, evaluate, and deploy mechanisms and strategies that are diverse and that continually shift and change over time to increase complexity and cost for attackers, limit the exposure of vulnerabilities and opportunities for attack, and increase system resiliency".

MTD provides a way to make it more difficult for an attacker to exploit a vulnerable system. The idea is to change one or more properties of a system in order to present attackers with a varying *attack surface*, so that, by the time the attacker has gained enough information about the system for planning an attack, the system's attack surface will be different enough to disrupt it. According to the definition of [1], "A system's attack surface is the subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack". It depends on the system's HW and SW features, and can be changed by dynamically reconfiguring such characteristics, working at different levels of granularity.

As suggested by [8], MTD approaches (also referred to as diversity techniques) may be applied both at a low level (e.g. working on code location in memory) and at the application level. The advantage of applying low-level diversity is that it does not require the understanding of the application's behavior and can be done automatically, but it is only capable of thwarting specific classes of attacks, such as code injection and memory corruption attacks.

Several low level MTD techniques have been proposed in the literature, based on the idea of automatically generating diverse variants of a program to disrupt exploits (diversity in program execution), first introduced in [9].

A widely deployed example is Address Space Randomization, that was introduced in 2000 by the PAX Team for Linux [10], and has been implemented in most modern operating systems. The basic idea is to randomize the locations of objects in memory so that an attack depending on the knowledge about the address of these objects will fail.

Instruction Set Randomization [11] is another technique for obfuscating the language understood by a system to protect against code-injection attacks: by randomizing the underlying systems instructions, *foreign* code introduced by an attack would fail to execute correctly, regardless of the injection approach.

Another type of low-level diversification is altering how data is stored in memory: in [12] authors present Data Randomization, a technique that provides probabilistic protection against attacks that exploits memory errors by XOR-ing data with random masks. Data randomization uses static analysis to partition instruction operands into equivalence classes: it places two operands in the same class if they may refer to the same object in an execution that does not violate memory safety. Then it assigns a random mask to each class and it generates code instrumented to XOR data read from or written to memory with the mask of the memory operands class. Therefore, attacks that violate the results of the static analysis have unpredictable results.

Jackson et al. present in [13] a diversity technique based on the generation, during the compilation phase, of multiple functionally equivalent machine codes for the same high-level source: with massive-scale software diversity, every user could get its own diversified program version, so that it is impossible for attackers to run a successful attack.

Looking at higher-level MTD techniques, several approaches have been proposed, aimed at thwarting the reconnaissance effort of attackers: reconnaissance enables an adversary to gather information about network topology, network dynamics, and even critical system and service information of the target system. This information can be used to identify system vulnerabilities, and to design and execute specific exploits on the system or services.

In this regard, several approaches for dynamically changing nodes IP addresses for proactive security have been proposed in the literature. In 2001 Kewley et al. [14] presented a technique called DYNAT (Dynamic Network Address Translation), aimed at confusing any adversary sniffing the network by obfuscating host identity information in TCP/IP packet headers when packets enter public parts of the

network. Whenever a client host wants to communicate with a protected server host, the addressing information contained in the header of its request packets is translated (encrypted) by a DYNAT shim before routing the packet to the server. A server gateway receives the packets, reverses the translation in the header fields (decryption) and obtains the true host identity information, used to pass the packets to the target server. Both the client and the server gateway must share a secret seed value, that is used to encrypt the identity information at sender side and decrypt them at the recipient. They are synchronized to periodically change the secret, and thus change the translation results, making it difficult for the adversary to create and maintain a map of the network. Although this technique has the advantage of providing a transparent approach to protect node identities from sniffing, it has been designed to protect a set of static nodes deployed behind a centralized gateway, that represents an interface between the protected network and the external world and performs the translation of addressing information for all incoming and outgoing packets. When considering more complex scenarios, characterized by highly dynamic network configurations, this approach would not work as it might be impossible to manage all communications through the centralized gateway and achieve node synchronization.

Another work funded by DARPA is presented in [15] by Atighetchi et al., that give an overview of current set of network-level defenses in the DARPA APOD (Application That Participate in Their Own Defense) project. Among the proposed network-centric defense mechanisms, the APOD toolkit also provides a *port and address hopping mechanism*, based on constantly changing a service's TCP identity to both hide the service's real identity and confuse the attacker during reconnaissance. Packets intercepted by attackers will reveal random addresses, which are valid only for a small period of time, e.g., 1 minute. For a port attack to be successful, the attacker must discover the current ports and execute the attack all within one refresh cycle. Similarly to the previous described approach, the hopping mechanism is implemented by a client component, directly located on the client machine, that intercepts higher level calls to the real server, and replaces all (realaddress:realport) header information with (fakeaddress:fakeport). The NAT

gateway is located either on the servers LAN or directly on the server host and performs the reverse mapping from (fakeaddress:fakeport) to (realaddress:realport). Even if this approach provides better unpredictability of identities than DYNAT, it also requires synchronization among the two communicating components, and the same considerations previously made apply in this case.

Antonatos et al. [16] introduce a proactive defense mechanism called Network Address Space Randomization (NASR) whose objective is to harden networks against worms that use precomputed hitlists of vulnerable targets, by forcing nodes to frequently change their IP addresses. In order to achieve this goal, the authors implemented an advanced NASR-enabled DHCP server to expire DHCP leases at intervals suitable for effective randomization. As the addresses are actually changed at the end-points of a communication, active connections are disrupted during the update; moreover, NASR is limited in the address space as it uses LAN addresses, and requires changes to the end-host operating system, thus making the deployment costly.

In [17] the authors introduce an MTD technique called OpenFlow Random Host Mutation (OF-RHM): each host is assigned an address range, selected from the entire unused address space in the network, and at each mutation interval, a virtual IP is chosen from this range and associated with the host. A Software-Defined Networking (SDN) approach is adopted for range allocation and mutation coordination: a centralized controller (NOX) properly installs flows in OpenFlow switches to forward requests and perform the address translation actions.

If considering some of the most adopted security solutions for networked systems, a common MTD practice consists in updating the cryptographic keys used for encryption of communication channels; this introduces some uncertainty for attackers but presents the problem of key distribution, that is a critical phase particularly subject to attacks.

As shown in details in the following section, in this thesis we propose an MTD-inspired framework, based on reconfiguration at different levels, with the reconfiguration granularity chosen at runtime based on current requirements. We focus

on physical reconfiguration, consisting in actually changing some of the system's parameters; by the way, virtual reconfiguration strategies aimed at presenting attackers with a virtual view of the system (about nodes ID, topology, traffic pattern) can be devised, and easily plugged into our framework.

2.2 The proposed MTD approach

As anticipated, an intuitive means to implement MTD is *reconfiguration*, that could be performed both at a physical level, that is by actually changing some system parameters (firmware, implemented cryptosystem, message format, etc.), and at a virtual level, that is by adopting mechanisms to give the attackers a virtual view of the system that does not correspond to the real one. We introduced an MTD strategy based on fine-grained physical reconfiguration, that takes into account the specific security and performance requirements depending on the deployment scenario while having in mind the HW and SW features of the nodes.

By reconfiguring a system, it is possible to increase the overall *level of security* provided by the system, both pro-actively, by periodically switching to a new configuration to reduce the time each system configuration is exposed to malicious observers, and re-actively, by scheduling a new configuration either after some detection event (e.g. in order to cope with specific attack typologies), or to meet new security requirements. Moreover, reconfiguration should take into account the current resource consumption and elaboration time, evaluating if and how this negatively affects the security level.

A reconfiguration approach can be formalized by defining the following items:

- **Reconfiguration Model**: identification of the system's reconfigurable parameters and of the admissible configurations;
- **Reconfiguration Strategies**: scheduling of the new reconfiguration based on the security and performance requirements;

- **Reconfiguration Mechanisms**: enforcement of the new configuration, through proper mechanisms.

In order to refer our examples to specific physical parameters for reconfiguration and specific mechanisms, in our discussion we will explicitly refer to a network of embedded nodes (e.g. WSNs, Smartphones, FPGA). Embedded nodes are systems designed to perform specific functions, that can be interconnected in order to achieve greater system design flexibility, improve diagnosability, and reduce wiring cost. In the following subsections, we present a reconfiguration model for a generic embedded node, and discuss about the level of security provided by a configuration and its dependency on time.

2.2.1 The reference reconfiguration model

An embedded node could be considered as structured into several architectural layers, as shown in Figure2.1.

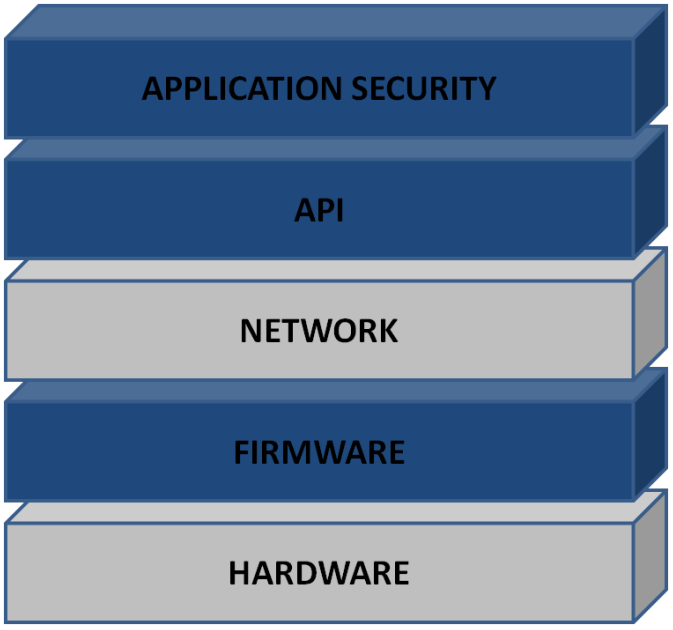

FIGURE 2.1: A layered view of an embedded node

Among them, we identified three main reconfigurable layers:

- *Security layer*. Application security in an embedded network can be achieved by implementing a proper cryptosystem to secure data exchanged among

nodes; security-level reconfiguration could be performed by switching among different cryptosystems, that satisfy specific security requirements while meeting certain performance and energy consumption constraints.

- *Application layer*. In order to perform complex tasks, embedded nodes communicate with one another according to specific application interfaces (APIs), defining the format of the exchanged messages and the communication protocols. Reconfiguration could be applied at this level by providing different APIs for the same application.

- *Physical layer*. In embedded systems, the software is embedded in the node firmware, that is typically preloaded on internal read-only memory (ROM) chip, in contrast to a general-purpose computer that loads its programs into random access memory (RAM) at run-time. Firmware provides the control program of the device and represents the skeleton where the different libraries for the implementation of the available cryptosystems and APIs can be plugged and activated via proper software switches. Nodes could be equipped with several versions of the firmware in order to perform physical reconfiguration when needed.

Clearly, further parameters could be considered for reconfiguration, such as the hardware configuration or the topology, as long as their reconfiguration is feasible from a technical and energy consumption point of view. Hardware reconfiguration is expensive and not feasible on most of the available devices but needed in case of damage. Network topology could be reconfigured in terms of the view offered to external observers; this could be achieved by implementing a mechanism that, for instance, presents virtual identities or introduces additional fake nodes into the network. Such a mechanism would need additional protocols and algorithms that are often too expensive for the considered nodes.

Consider an embedded network composed of N nodes. Let IM be the set of firmware versions available, API be the set of possible application interfaces, and

SEC be the set of available security mechanisms. The set of possible node configurations is defined as:

$$C_{node} = IM \times API \times SEC \tag{2.1}$$

The set of possible network configurations is then defined as a subset of the N-ary Cartesian power of the set C_{node}.

$$C_{net} \subseteq C_{node} \times C_{node} \cdots \times C_{node} = C_{node}^{N} \tag{2.2}$$

The choice of the reconfiguration level impacts both the system performance and the provided level of security. From the performance point of view, changing the firmware of the whole network or of a subset of it, is much more expensive, in terms of latency and power consumption, than changing the APIs or the cryptosystem, whose reconfiguration could be handled in software. On the other side, by changing the entire application running on a node, it becomes harder for an attacker to exploit software vulnerabilities in order to gain complete control of the node. Similarly, APIs' reconfiguration could be useful to confuse an attacker that is observing the communication protocol in order to find an exploit to disturb or control the communication.

At the security layer, the cryptosystem itself is designed to cope with a specific set of attacks and provides an *intrinsic level of security*, depending on the cryptographic scheme, the algorithm, the length of the keys, etc. Cryptosystem's reconfiguration can increase the level of security in two ways, that is:

- by switching to a cryptosystem that has itself a higher intrinsic level of security, absolutely or with respect to a specific set of attacks (that have been currently detected for example, and were not covered by the previous configuration), or

- by selecting an equivalent cryptosystem that uses different parameters – e.g. by updating the cryptographic key while keeping the same cryptographic algorithm.

Given a certain configuration, the more an attacker is able to observe, the more she will be able to infer about the system; by continuously changing the system configuration, the attacker will be presented with different view of the system over time, and will have to start from scratch many times to find an exploit.

While the choice of the firmware does not influence the way nodes communicate, cryptosystem and APIs reconfiguration has a direct impact on the format of exchanged packets and communication protocols. For this reason, some combinations of node configurations may not be valid as they would interfere with normal network operation. In order to preserve communication, a proper mechanism must be designed to ensure consistency among legitimate nodes through reconfigurations.

Once the reconfiguration model has been defined, it is possible to design a reconfiguration strategy able to select the reconfiguration granularity and the specific configuration to activate depending on given quality requirements. In the following sections we discuss about security level evaluation and security metrics, that constitute the fundamental parameter considered by the reconfiguration strategy.

2.2.2 Security Level Evaluation

In this section, we will formalize the notion of level of security associated with a node/network configuration, and will show its dependency on time. Based on the analysis of security and efficiency requirements, we will later define the reconfiguration function, aimed at preserving or increasing the level of security provided by a node or a network.

The security of complex systems depends on many technical and organizational issues that must be properly addressed. The need for a clear definition and selection of security rules has led system administrators to set up security policies trying to adopt formal approaches to describe system security configurations. In

spite of the ambiguity of such policies, a common approach to evaluate a system's security is through evaluation of its security policy. At present, such an evaluation is performed *by hand* whenever enterprises endeavor to extend their trusted domain and cooperate [18].

This approach also includes the well known standards as Common Criteria and TCSEC [19, 20], that are very suitable to assess and audit the security level provided by a company, by a specific procedure or, in general, by a system. The Common Criteria (CC) for Information Technology Security Evaluation (Common Criteria or CC) [19] are an internationally approved set of standard for computer security certification. They are used by Government customers in the USA and the NATO community along with other organizations, particularly in the public sector, to determine the level of security and assurance of various technology products. However, the assurance levels provided by CC (from EAL1 to EAL7) do not measure the security of the system itself, but simply state at what level the system was tested, and do not find a direct application in our approach.

Defining a quantitative measure of the level of security provided by a system is a complex task. Several security metrics have been proposed in literature, mostly based on the analysis of attack graphs or on risk quantification.

In [21], the authors present a stochastic model for quantifying security of networked systems. A vulnerability graph is used to abstract a networked system: a vertex may represent a vulnerability or a system with possibly multiple vulnerabilities, and an edge captures the relation that the exploitation of one vulnerability could lead to the exploitation of the other. A stochastic process (specifically, a renewal process) is used to model the evolution of a randomly picked vertex: each cycle of the renewal process is composed of the time interval corresponding to the secure state, and the time interval corresponding to the compromised state. The security metric used is "'the probability that a randomly picked vertex is compromised when the system enters its steady state"'; the authors aim to capture the impact of the state of the neighbors of a node on its own state, because a node may get compromised through an attack that is launched from one or multiple neighbors.

Barth et al. in [22] study the efficacy of security reactive strategies, considering a game-theoretic model with a strategic attacker who responds to the defenders strategy. They make worst case assumptions about the attacker (she holds complete knowledge of the defenders strategy and is not required to act rationally), and assume that the defender can observe the attackers past actions. Authors evaluate the defenders strategy by measuring the attackers cumulative return-on-investment, the return-on-attack (ROA). The focus is on defenders who seek to reduce the attackers incentives for attacking the enterprise. Authors compare the payoff the attacker receives from her attack with the cost of performing it.

The Common Vulnerability Scoring System (CVSS)[23] is a widely used and well-established standard for classifying the severity of security vulnerabilities. It provides a universal open and standardized method for rating IT vulnerabilities: CVSS consists of 3 groups of metrics: Base, Temporal and Environmental. Each group produces a numeric score ranging from 0 to 10, and a Vector, a compressed textual representation that reflects the values used to derive the score. The Base group represents the intrinsic qualities of a vulnerability. The Temporal group reflects the characteristics of a vulnerability that change over time. The Environmental group represents the characteristics of a vulnerability that are unique to any user's environment. A final score is computed by combining the score of each group. Generally, the base and temporal metrics are specified by vulnerability bulletin analysts, security product vendors, or application vendors because they typically have better information about the characteristics of a vulnerability than do users. The environmental metrics, however, are specified by users because they are best able to assess the potential impact of a vulnerability within their own environments.

In [24] Ahmed et al. propose a novel security metric framework that identifies and quantifies objectively the most significant security risk factors, which include existing vulnerabilities, historical trend of vulnerability of the remotely accessible services, prediction of potential vulnerabilities for any general network service and their estimated severity and finally policy resistance to attack propagation within the network. Security is measured based on two critical risk aspects - the risk of

having a successful attack and the risk of this attack being propagated within the network.

Finally, Jajodia et al. present in [25] a novel quantitative metric for the security of computer networks that is based on an analysis of attack graphs. The metric measures the security strength of a network in terms of the strength of the weakest adversary who can successfully penetrate the network. They present an algorithm that computes the minimal sets of required initial attributes for the weakest adversary to possess in order to successfully compromise a network, given a specific network configuration, set of known exploits, a specific goal state, and an attacker class.

All the above metrics are generally hard to link to the adopted definition of configuration, that is centered on the mechanisms that are available to enforce a subset of security requirements. For this reason, we adopt a different metric, based on the coverage of a set of known attacks.

An attack could have several objectives, such as physically taking possession of a node, interfering with communication at the physical level, exploiting software vulnerabilities to take control of a node, disturb network operation at routing/application level or intercept sensitive data. In this discussion we are interested in attacks aimed at interfering, steering or eavesdropping communications at the application layer among nodes, and at exploiting vulnerabilities of the firmware installed on nodes.

Let $Threats$ define the set of threats of interest, belonging to the above discussed set of attacks. A configuration c is said to *cover* a threat $t \in Threats$, if either the cryptosystem implemented at the security layer or the specific firmware version running on the node include mechanisms to protect the node from such threat.

Once the admissible configurations and the attacks of interest have been identified, it is possible to build an *Attack Coverage Table*, that helps define the levels of security provided by each configuration [26]. Table 2.1 shows an example of attack coverage table relative to configurations $\{c_1, c_2, c_3, c_4\}$, under the hypothesis that three attacks of interest have been identified, namely $AttackA$, $AttackB$ and

Conf	*Attack A*	*Attack B*	*Attack C*	*SL*
c_1	×			L_1
c_2	×		×	L_2
c_3	×	×	×	L_3
c_4	×		×	L_2

TABLE 2.1: An example of Attacks Coverage Table

AttackC. An increasing level of security (from L_1 to L_4 in the example) can be assigned to configurations, based on the risk associated with the attacks and their coverage properties.

Coverage can be defined either as a ON/OFF property (that is an attack is covered or uncovered), or in terms of the degree of satisfaction of specific requirements (e.g. authentication, integrity, confidentiality, key distribution...), using a scoring system (similar to CVSS for vulnerabilities).

The level of security associated with a configuration could simply depend on the number of covered threats, or it could be set depending on the risk associated with each threat, either in a static way (the risk associated with a threat is set at deployment and remains unchanged for the entire operation of the network) or dynamically (the risk associated with a threat changes dynamically during network operation depending on current conditions and possible detection events).

2.2.3 Security level dependency on time

As already said, the level of security provided by a configuration depends on the implemented cryptosystem (cryptographic scheme, algorithms and keys) at the security layer and the installed firmware version at the physical layer, and is characterized by an intrinsic value that quantifies the effort needed by an attacker to break it. Indeed, the more a system configuration is exposed to malicious observers, the more the actual level of security decreases; for this reason, the security level is a monotonically decreasing function of time, having its maximum in the intrinsic value associated with the particular configuration.

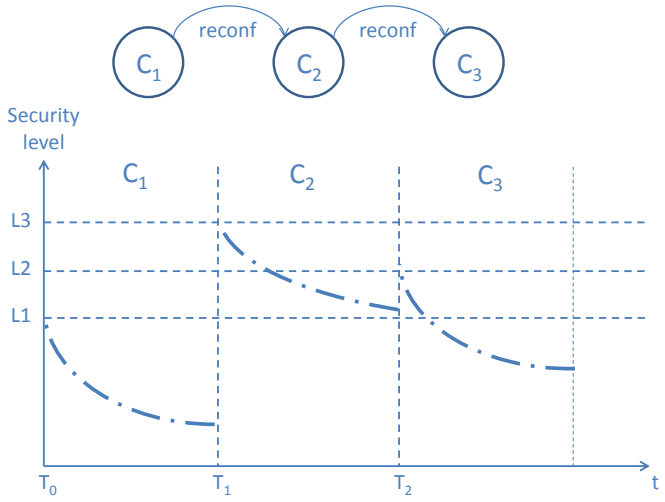

FIGURE 2.2: Level of security through reconfigurations

As graphically illustrated in Figure 2.2, thanks to reconfiguration, it is possible to avoid that the level of security goes below a certain threshold, and periodically re-start from the intrinsic value associated with the new configuration. Dually, thanks to reconfiguration, we avoid that the probability of successfully completing an attack increases. In fact, such probability depends on the considered type of attack and is represented by an increasing function: the longer an attacker can try to exploit a system, the higher the probability of success. Clearly, if the attacker has unlimited time, eventually she will be able to break the system. By introducing reconfiguration, as we will theoretically prove in section 5.3, the probability of performing a successful attack is decreased, as the attacker's effort is nullified each time a new configuration is activated.

In the following, we will refer to the *level of security* as a security metric to express how secure a configuration is with respect to the considered attacks. A security value can be assigned, based on the attacks coverage table, both to a single node and to a link, defined as a connection between communicating nodes. Node security is related primarily to the physical layer (e.g., tamper resistant HW, protected external ROM), while subnet security depends on the security layer (e.g., cryptographic algorithm, key length, key agreement mechanisms); as previously discussed, both also depend on the reconfiguration mechanism itself, that is on time.

Assume that the set of available cryptosystems is a totally ordered set: given s_1,$s_2 \in SEC$, there is an ordering relation between them, and $s_1 \leq s_2$ means that the cryptosystem s_1 is not more secure than the cryptosystem s_2. It is possible to have elements in SEC that are equivalent from the security point of view, adopting for instance the same algorithm but using different parameters (e.g. different keys).

Assume the sequence of the M configurations adopted by a node n is given by $\langle C_1(n), \ldots, C_M(n)\rangle$, and the sequence of time instants in which such configuration were activated is $\langle T_1(n), \ldots, T_M(n)\rangle$. Let the configurations $C_i(p) = (im_i(p), api_i(p), s_i(p))$ and $C_i(q) = (im_i(q),\ api_i(q), s_i(q))$ be the i-th active configurations respectively on node p and q. Note that in order for the nodes to be able to communicate, they should either share the same security and API configurations, or they should be provided with a mechanism to always know what is the configuration currently used by other legitimate nodes. Let $T_i(p, q)$ identify the initial time instant when the status of p and q is such that they are able to communicate.

In the following, we will refer to a *link* as a directed edge (p, q) connecting two nodes involved in a communication, with packets traveling from p to q. A link configuration is defined as $C_i(p, q) = (C_i(p), C_i(q))$.

Let us refer to $SL_{(p,q)}(t)$ as the level of security, at time t, of a link (p, q). It is the level of security associated with the cryptosystem used to secure data flow from p to q, denoted with $s_i(p, q)$. With $SL_p(t)$ we identify the level of security of node p, depending on its physical configuration $im_i(p)$ and on time.

Definition 2.1 (Level of security of a link)**.** According to the previous considerations, the level of security $SL_{(p,q)}(t)$ of a link (p, q), provided by $C_i(p, q)$ at time $t \in [T_i(p, q); T_{i+1}(p, q)]$, can be expressed as a function of the specific cryptosystem adopted $s_i(p, q)$ and the time elapsed since the current configuration was activated.

$$SL_{(p,q)}(t) = f(s_i(p, q), t - T_i(p, q)) \tag{2.3}$$

Definition 2.2 (Level of security of a node)**.** Similarly, the level of security $SL_p(t)$ of a node p can be expressed as a function of the specific firmware adopted $im_i(p)$

and the time elapsed since the current physical configuration was activated.

$$SL_p(t) = f(im_i(p), t - T_i(p)) \tag{2.4}$$

Definition 2.3 (Level of security of the network)**.** Assuming that the network is partitioned in different subnets, each composed of nodes communicating with one another with a certain interface (security and application layer), the overall level of security of the network depends both on the security of nodes composing the network, and of the different subnets, other than on time.

$$SL_{net}(t) = A \cdot \sum_{i=0}^{N-1} \sum_{j=0, j\neq i}^{N-1} \alpha_{ij} \cdot SL_{(i,j)}(t) \cdot x_{ij} + B \cdot \sum_{i=0}^{N-1} \beta_i \cdot SL_i(t) \tag{2.5}$$

A and B respectively represent the relative importance of the set of links and the set of network nodes respectively, and satisfy the following constraint: $A + B = 1$.

The α_{ij} represent link weights, while β_i are node weights and are useful to give more importance to critical nodes or portions of the network. They are subject to the following constraints:

$$\sum_{i=0}^{N-1} \sum_{j=0, j\neq i}^{N-1} \alpha_{ij} = 1 \quad \text{and} \quad \sum_{i=0}^{N-1} \beta_i = 1 \tag{2.6}$$

The x_{ij} variables represent the existence of links and are defined as follows.

$$x_{ij} = \left\{ \begin{array}{ll} 1 & if \quad i \neq j \text{ and } \exists \text{ a link between node } i \text{ and } j \\ 0 & if \quad i \neq j \text{ and } \nexists \text{ a link between node } i \text{ and } j \end{array} \right\} \tag{2.7}$$

2.2.4 Reconfiguration Strategies

Once the notion of level of security associated with a configuration has been formalized, it is possible to define the reconfiguration strategy, that depends not only on specific security requirements, but also on additional performance parameters.

The selection of the new configuration to activate is performed by a security driven scheduler. The scheduler can be either a centralized entity making decisions on the global network configuration, or a decentralized component, independently deployed on each network node, making local reconfiguration decisions.

In a *centralized* approach, a central entity triggers a configuration update based on some events – e.g., timer expiration, detected security threat – and transmits its decision to all nodes involved in communication.

In a *de-centralized* approach, each node is able to schedule, independently from other nodes, when to update its own configuration. Communication among legitimate nodes is preserved adopting additional mechanisms; we will give some details about a possible reconfiguration protocol in the following chapters.

A local or global reconfiguration task can be triggered either periodically or after the detection of an attack. In the second case, the detection of an ongoing attack could create new security requirements, thus influencing the selection of the new configuration. Second, each reconfiguration task is associated with a cost, representing the effort needed to switch to the new configuration, in terms of both energy consumption and delays introduced into network operation.

The current battery level is a fundamental parameter to decide about the new configuration: for example, if the node battery is low, a reconfiguration may not be possible in practice. Let $Battery = \{lo, med, hi\}$ be the set of possible battery levels of a node. As for performance, the distributed application running on the network may have specific requirements with respect to the delays that can be tolerated when switching among different configurations. $Delay = \{restart\_tolerated, restart\_not\_tolerated\}$ is an example of the delay allowance levels.

The node reconfiguration function can be denoted as:

$$node\_reconf : C_{node} \times L \times L \times Battery \times Delay \times Threats \rightarrow C_{node} \qquad (2.8)$$

where the input arguments are respectively: the current configuration $c^{curr}(i) \in C_{node}$, the current level of security $L^{curr}(i) \in L$, the desired level of security

$L^{new}(i) \in L$, the current battery level $b(i)$
$inBattery$, an indication of the delay allowance level $d(i) \in Delay$, and a detected alert $threat(i) \in Threats$.

The network reconfiguration function is similarly defined as:

$$net\_reconf : C_{net} \times L \times L \times Delay \rightarrow C_{net} \tag{2.9}$$

Chapter 3

A case study: WSN security

In this chapter, we address the main security issues arising in networks composed of resource-constrained devices, showing that providing high levels of security in those networks is a complex and challenging task, due to the inescapable trade-off among security and performance. Even if most of the considerations apply to a wide range of constrained architectures, in order to be able to refer to real applications, we will explicitly refer to Wireless Sensor Networks. In this chapter in particular, the peculiar HW/SW features of sensor nodes are described, highlighting the main consequent challenges for security and performance, and the most common attacks against WSNs are presented, along with some of the most relevant security solutions proposed in the literature.

3.1 Wireless Sensor Networks: an overview

Wireless Sensor Networks (WSNs) are usually composed of several sensor nodes (also called *motes*), typically self-powered and provided with simple sensing and forward capabilities, and one or more base stations (BS), often represented by more powerful devices that act as gateways towards the external world. A high-level monitoring application usually interacts with the BSs, by sending them commands or queries, that are spread into the network through radio links.

A sensor node is composed of several parts: a *radio transceiver* with an internal antenna or connection to an external antenna, a *microcontroller*, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting.

TinyOS [27] is the most commonly adopted operating system for WSNs. TinyOS applications and the OS itself are built by connecting so called components, that represent functional building blocks such as communication protocols, device drivers, or data analysis modules. During the default compilation process of TinyOS, these building blocks are converted into a monolithic, static binary, to enable code optimization and ensure a small memory footprint.

Sensor nodes typically have a small form factor with a limited amount of battery power, are equipped with small programming memories and microprocessor boards with limited computational capabilities. The communication range of sensor nodes is also limited, both because of technical constraints and by the need to conserve energy; sensor nodes are prone to failure due to loss of power, and are often left unattended because they are typically deployed in hostile and harmful environments.

A typical monitoring system is made of different sensor networks that can be heterogeneous in the technology aspects, in the data formats, in synchronization and localization standards and so on. They can be connected in different ways and their data are elaborated by the same application to enrich the knowledge of observed complex phenomena. As illustrated in Figure3.1, such an architecture can be considered as structured into two main layers, namely the *sensor network layer* and the *distributed application layer*.

The *sensor network layer* can be further divided into two levels:

- Physical level: is responsible of the processing of the locally generated data at the node level.
- Transport level: controls the communication between the nodes of the network.

The *application layer* deals with the fusion and high level management of the data sensed by the different heterogeneous networks; it can be considered as structured into two levels:

- Integration level: is responsible of the integration of data belonging to different sensor networks; it typically enforces a translation in a common data model.
- User level: executes the user distributed applications, which typically query the underlying networks and sensor features and manipulate the retrieved results for aggregation and decision purposes.

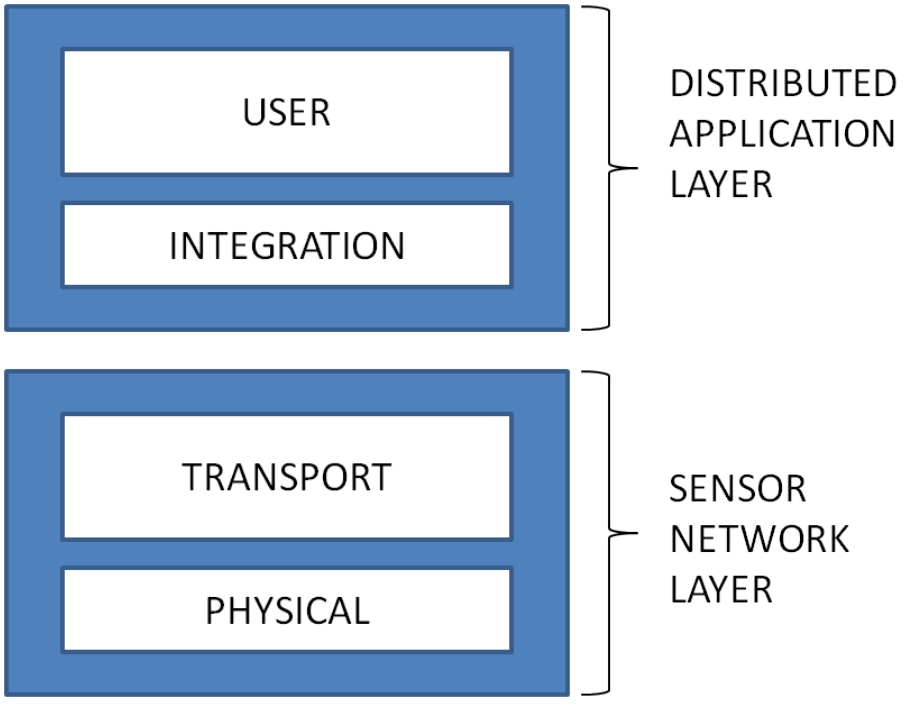

FIGURE 3.1: A monitoring system layered view

3.2 WSN security issues

WSNs are widely used in several application domains as environmental monitoring, detection and classification of objects in military and civil settings, agriculture procedures, automotive and health monitoring. Their decentralized and self-organizing nature makes the deployment very easy and this facilitates their

adoption in any context without requiring the existence of any supporting infrastructure. Furthermore, they are widely employed in critical scenarios and security issues are becoming a fundamental concern to be addressed by means of proper security policies and mechanisms.

Referring to the Figure3.1 in the previous section, security issues arise at all the architectural levels depicted in it: at the *application layer*, data retrieved from the different networks are typically processed in a distributed manner, thus raising well-known issues dealing with secure network communication and access control; as this kind of processing is usually done by PC-class devices, the application layer does not suffer of the problems related to the limited resources of sensor nodes, and the well-known security protocols can be directly applied.

As for the *sensor network layer*: at the *transport level* it is necessary to secure data exchanged between nodes, and this can be achieved with the adoption of proper security protocols and mechanisms that take in consideration the limited resources; at the *physical level*, it is necessary to provide mechanisms for protecting nodes against physical tampering and DOS and jamming attacks.

3.3 Security vulnerabilities in WSNs

The wireless nature of WSNs communications makes it possible to wage different types of attacks ranging from passive eavesdropping to active interfering. In this section we present a classification of WSN security threats, based on recent surveys [28–30].

3.3.1 Attacks classification based on attacker capabilities

External Vs. Internal. External attacks are waged by nodes that do not belong to the WSN, while internal attacks occur when legitimate nodes of a WSN are compromised and behave in unintended or unauthorized ways.

Mote-class Vs. Laptop-class. In mote-class attacks, an adversary attacks a WSN by using a few nodes with similar capabilities to the network nodes; in laptop-class attacks, an adversary can use more powerful devices (e.g., a laptop) to attack a WSN. These devices have greater transmission range, processing power, and energy reserves than the network nodes.

3.3.2 Attacks classification based on attackers goals

Eavesdropping. In a sensor network, the base station typically sends commands and queries to sensors, that monitor specific physical phenomena and report to the base station accordingly. Eavesdropping threatens confidentiality as it aims at gaining unauthorized access to data contained in exchanged packets.

Node capture. This attack aims at compromising a network node by tampering with the hardware to extract the program code, data and keys stored within a sensor node, or by attempting to load the attacker program in the compromised node. It also involves breaking the software running on the sensor nodes. Once a node is captured, the attacker becomes an insider and can use the node to perform internal attacks.

Interruption. This attack threatens service availability, as it aims at making communication links in sensor networks become lost or unavailable. The main purpose is to launch denial-of-service (DoS) attacks.

Modification. An unauthorized party not only accesses the data but also tampers with it. This threatens message integrity. The main purpose is to confuse or mislead the parties involved in the communication protocol.

Fabrication. An adversary injects false data and compromises the trustworthiness of information. This threatens message authenticity. The main purpose is to confuse or mislead the parties involved in the communication protocol. This operation can also facilitate DOS attacks, by flooding the network.

Replay existing messages. This operation threatens message freshness. Again, the main purpose of this operation is to confuse or mislead the parties involved in the communication protocol that is not time-aware.

Protocol-specific compromise. This includes all the attacks on information in transit, and also includes deviating from protocols: when the attacker is, or becomes an insider of the network, and the attackers purpose is not to threaten the service availability, message confidentiality, integrity and authenticity of the network, but to gain an unfair advantage for itself in the usage of the network, the attacker manifests selfish behaviors, behaviors that deviate from the intended functioning of the protocol.

3.3.3 Attacks classification based on protocol stack

WSN attacks can be classified based on the layer in the protocol stack that they are targeted at. As classical wireless networks, WSN architectures follow the OSI Model, consisting in five layers: application layer, transport layer, network layer, data link layer and physical layer.

3.3.3.1 Physical Layer

The physical layer is responsible for frequency selection, carrier frequency generation, signal detection, modulation, and data encryption. Sensor networks typically operate in hostile outdoor environments. In such environments, the small form factor of the sensors, coupled with the unattended and distributed nature of their deployment make them highly susceptible to physical attacks, i.e., threats due to physical node destructions.

Attackers can extract cryptographic secrets, tamper with the associated circuitry, modify programming in the sensors, or replace them with malicious sensors under the control of the attacker.

Jamming: This is one of the Denial of Service attacks in which the adversary attempts to disrupt the operation of the network by broadcasting a high-energy

signal. Jamming attacks in WSNs can be waged in different ways, that is by corrupting packets as they are transmitted, sending a constant stream of bytes into the network to make it look like legitimate traffic, randomly alternating between sleep and jamming to save energy, and transmitting a jam signal when it senses traffic.

There are several attack techniques:

- Spot Jamming: the attacker directs all its transmitting power on a single frequency that the target uses with the same modulation and enough power to override the original signal. Spot jamming is usually very powerful, but since it jams a single frequency each time it may be easily avoided by changing to another frequency.
- Sweep Jamming : a jammer shifts rapidly from one frequency to another. While this method of jamming has the advantage of being able to jam multiple frequencies in quick succession, it does not affect them all at the same time, and thus its effectiveness is limited.
- Barrage Jamming : a range of frequencies is jammed at the same time. Its main advantage is that it is able to jam multiple frequencies at once with enough power to decrease the SNR of the enemy receivers. However as the range of the jammed frequencies grows bigger the output power of the jamming is reduced proportionally.

Algorithms that combine statistically analyzing the received signal strength indicator (RSSI) values, the average time required to sense an idle channel (carrier sense time), and the packet delivery ratio (PDR) techniques can reliably identify jamming. As a defense against jamming attack, spread-spectrum techniques for radio communication are adopted.

Radio interference. In this type of attack the adversary produces large amounts of interference either intermittently or persistently. To handle this issue it is possible to use symmetric key algorithms in which the disclosure of the keys is delayed by some time interval.

Tampering or destruction. Given physical access to a node, an attacker can extract sensitive information such as cryptographic keys or other data on the node. One defense to this attack involves tamper-proofing the nodes physical package, adopting for example Self Destruction (tamper-proofing packages) devices. In such devices, whenever somebody accesses the sensor nodes physically the nodes vaporize their memory contents and this prevents any leakage of information.

3.3.3.2 Data Link Layer

The link layer is responsible for multiplexing of data-streams, data frame detection, medium access control, and error control. Attacks at this layer include purposefully created collisions, resource exhaustion, and unfairness in allocation.

Continuous Channel Access (Exhaustion). A malicious node disrupts the Media Access Control protocol, by continuously requesting or transmitting over the channel. This eventually leads a starvation for other nodes in the network with respect to channel access. One of the countermeasures to such an attack is Rate Limiting to the MAC admission control, such that the network can ignore excessive requests, thus preventing the energy drain caused by repeated transmissions. A second technique is to use time-division multiplexing where each node is allocated a time slot in which it can transmit.

Collision. This is very much similar to the continuous channel attack. A collision occurs when two nodes attempt to transmit on the same frequency simultaneously. When packets collide, a change will likely occur in the data portion, causing a checksum mismatch at the receiving end. The packet will then be discarded as invalid. A typical defense against collisions is the use of error-correcting codes.

Unfairness. Repeated application of these exhaustion or collision based MAC layer attacks or an abusive use of cooperative MAC layer priority mechanisms, can lead into unfairness. This kind of attack is a partial DOS attack, but results in marginal performance degradation. A defensive measure against such attacks is the usage of small frames, so that any individual node seizes the channel for a smaller duration only.

Interrogation. Exploits the two-way request-to-send/clear to send (RTS/CTS) handshake that many MAC protocols use to mitigate the hidden-node problem. An attacker can exhaust a nodes resources by repeatedly sending RTS messages to elicit CTS responses from a targeted neighbor node. To thwart such type of attacks, a node can limit itself in accepting connections from same identity or use anti-replay protection and strong link-layer authentication.

Sybil Attack. In this attack a single malicious node assumes several identities, known as Sybil nodes. Many MAC protocols adopt voting for finding the better link for transmission from a pool of available links. The Sybil Attack could be used to steer the voting process.

3.3.3.3 Network Layer

The major function of this layer is routing. This layer presents many challenges, due to limited power, memory and computational capability resources, and is subject to several attacks.

Sinkhole. In a Sinkhole attack a compromised node tries to draw all or as much as possible traffic from a particular area, by making itself look attractive to the surrounding nodes with respect to the routing metric. As a result, the adversary manages to attract all traffic that is destined to the base station. By taking part in the routing process, she can then launch more severe attacks, like selectively forwarding, modifying or even dropping the packets coming through.

Hello Flood. This attack exploits Hello packets that are required in many protocols to announce nodes to their neighbors. A node receiving such packets may assume that it is in radio range of the sender. A laptop-class adversary can send this kind of packet to all sensor nodes in the network, so that they believe the compromised node belongs to their neighbors. This causes a large number of nodes sending packets to this imaginary neighbor and thus into oblivion. Authentication is the key solution to such attacks.

Selective Forwarding / Black Hole Attack. WSNs are usually multi-hop networks and hence based on the assumption that the participating nodes will forward the messages faithfully. Malicious or attacking nodes can however refuse to route certain messages and drop them. If they drop all the packets through them, then it is called a Black Hole Attack. However if they selectively forward the packets, then it is called selective forwarding. To overcome this, Multi-path routing can be used in combination with random selection of paths to destination, or braided paths can be used which represent paths which have no common link or which do not have two consecutive common nodes, or use implicit acknowledgments, which ensure that packets are forwarded as they were sent.

Sybil Attack. A previously said, in this attack, a single node presents multiple identities to all other nodes in the WSN. This may mislead other nodes, and hence routes believed to be disjoint with respect to node can have the same adversary node. A countermeasure to Sybil Attack is by using a unique shared symmetric key for each node with the base station.

Wormhole Attacks. An adversary can tunnel messages received in one part of the network over a low latency link and replay them in another part of the network. This is usually done with the coordination of two adversary nodes, where the nodes try to understate their distance from each other, by broadcasting packets along an out-of-bound channel available only to the attacker. To overcome this, the traffic can be routed to the base station along the geographically shortest path, or it is possible to use very tight time synchronization among the nodes, which is infeasible in practical environments.

Spoofed, Altered, or Replayed Routing Information. The most direct attack against a routing protocol in any network is to target the routing information itself while it is being exchanged between nodes. An attacker may spoof, alter, or replay routing information in order to disrupt traffic in the network. These disruptions include the creation of routing loops, attracting or repelling network traffic from select nodes, extending and shortening source routes, generating fake error messages, partitioning the network, and increasing end-to-end latency. A

countermeasure against spoofing and alteration is to append a message authentication code (MAC) after the message. Efficient encryption and authentication techniques can defeat spoofing attacks.

Acknowledgment Spoofing. Routing algorithms used in sensor networks sometimes require Acknowledgments to be used. An attacking node can spoof the Acknowledgments of overheard packets destined to neighboring nodes in order to provide false information to those neighboring nodes. The most obvious solution to this problem would be authentication via encryption of all sent packets and also packet headers.

Misdirection. This is a more active attack in which a malicious node present in the routing path can send the packets in wrong directions through which the destination is unreachable. Instead of sending the packets in correct direction the attacker misdirects them towards one node that can thus become the victim of a DOS attack. If it gets observed that a node's network link is getting flooded without any useful information then the victim node can be scheduled into sleep mode for some time to over come this.

Internet Smurf Attack. In this type of attack the adversary can flood the victim node's network link. The attacker forges the victim's address and broadcasts echoes in the network and also routes all the replies to the victim node. This way the attacker can flood the network link of the victim. If it gets observed that a node's network link is getting flooded without any useful information then the victim node can be scheduled into sleep mode for some time to over come this.

Homing. It uses traffic pattern analysis to identify and target nodes that have special responsibilities, such as cluster heads or cryptographic-key managers. An attacker then achieves DoS by jamming or destroying these key network nodes. Header encryption is a common prevention technique. Using dummy packets throughout the network to equalize traffic volume and thus prevent traffic analysis. Unfortunately, this wastes significant sensor node energy, so it can be sued only when preventing traffic analysis is of utmost importance.

3.3.3.4 Transport Layer

The function of this layer is to provide reliability and congestion avoidance to the communication. The attacks that can be launched on the transport layer in a WSN are flooding attack and de-synchronization attack.

Flooding. An attacker may repeatedly make new connection requests until the resources required by each connection are exhausted or reach a maximum limit. One proposed solution to this problem is to require that each connecting client demonstrates its commitment to the connection by solving a puzzle. As a defense against this class of attack, a limit can be put on the number of connections from a particular node.

De-synchronization Attacks. In this attack, the adversary repeatedly forges messages to one or both end points which request transmission of missed frames. Hence, these messages are again transmitted and if the adversary maintains a proper timing, it can prevent the end points from exchanging any useful information. This will cause a considerable drainage of energy of legitimate nodes in the network in an endless synchronization-recovery protocol. A possible solution to this type of attack is to require authentication of all packets including control fields communicated between hosts. Header or full packet authentication can defeat such an attack.

3.3.3.5 Application Layer

This layer is responsible for traffic management, and provides software for different applications that translate the data in an understandable form or send queries to obtain certain information.

Overwhelm attack. An attacker might attempt to overwhelm network nodes with sensor stimuli, causing the network to forward large volumes of traffic to a base station. This attack consumes network bandwidth and drains node energy. We can mitigate this attack by carefully tuning sensors so that only the specifically desired stimulus, such as vehicular movement, as opposed to any movement,

triggers them. Rate-limiting and efficient data-aggregation algorithms can also reduce these attacks effects.

Path-based DOS attack. It involves injecting spurious or replayed packets into the network at leaf nodes. This attack can starve the network of legitimate traffic, because it consumes resources on the path to the base station, thus preventing other nodes from sending data to the base station. Combining packet authentication and anti-replay protection prevents these attacks.

Deluge (reprogram) attack. Network-programming system let you remotely reprogram nodes in deployed networks If the reprogramming process is not secure, an intruder can hijack this process and take control of large portions of a network.

3.4 Existing approaches to securing a WSN

As previously said, WSN nodes are typically provided with constrained processing and storage capabilities and limited energy resources; they are prone to failures due to harsh deployment environments and are easy to be compromised due to typically unattended operations. Finally, a WSN is often characterized by a dynamic topology due to node joining, mobility or failure, thus introducing further security and reliability issues. The specific features of the sensor nodes make difficult the direct application of the existing security approaches to the area of wireless sensor networks: most security protocols are based on cryptographic operations, which massively involve the adoption of keys and complex mathematical functions that require dedicated computational resources and turn out to be critical from a performance and power consumption point of view.

The security of a cryptographic system relies mainly on the secrecy of the key it uses. Keys for these cryptographic operations must be set up by communicating nodes before they can exchange information securely. Key management schemes are mechanisms used to establish and distribute various kinds of cryptographic keys in the network, such as individual keys, pair wise keys, and group keys.

Key management is an essential cryptographic primitive upon which other security primitives are built. If an attacker can find the key, the entire system is broken. In fact, a secure key management scheme is the prerequisite for the security of these primitives, and thus essential to achieving secure infrastructure in sensor networks. In sensor networks end-to-end encryption is impractical because of large number of communicating nodes and each node is incapable of storing large number of encryption keys. Therefore hop-by-hop encryption mechanism is usually used in which each sensor node stores only encryption keys shared with its immediate neighbors.

So, the main problem to face with when setting up a secure communication between nodes is the way cryptographic keys are established at each node. There are two main well-known mechanisms to handle this problem: in Symmetric Key Cryptography (SKC) a unique secret shared key is used for both encrypting and decrypting messages, while in Public Key Cryptography (PKC) each node manages a couple of keys.

The process by which public key and symmetric key cryptography schemes should be selected is based on the following criteria:

- Energy: how much energy is required to execute the encrypt/decryption functions
- Program memory: the memory required to store the encryption/decryption program
- Temporary memory: the required RAM size or number of registers required temporarily when the encryption/decryption code is being executed
- Execution time: the time required to execute the encryption/decryption code
- Program parameters memory: the required memory size to save the required number of keys used by the encryption/decryption function

Due to WSN nodes constraints, asymmetric cryptography is often too expensive for many applications. Thus, a promising approach is to use more efficient symmetric

cryptographic alternatives. However, symmetric cryptography is not as versatile as public key cryptographic techniques, which complicates the design of secure applications. Applying any encryption scheme requires transmission of extra bits, hence extra processing, memory and battery power, which are very important resources for the sensors longevity. Applying the security mechanisms such as encryption could also increase delay, jitter and packet loss in WSNs.

Several implementations of Symmetric Key Cryptography (SKC) algorithms in WSN have been proposed in literature (TinySec [31], MiniSec [32], ZigBee [33] and SNEP [34]), thanks to their low computational costs that make them well suited for realization on sensor devices. Even if symmetric schemes are very attractive for their energy and memory efficiency, they require complex and resource expensive key distribution and management protocols, resulting in a heavy traffic in the network and in complex and not scalable architectures. Moreover, symmetric cryptography only fulfills confidentiality requirements, while not considering other security issues such as authentication and integrity. As a matter of fact, an important security requirement which arises within the sensor network domain is the broadcast authentication, that is the capacity of a sender to broadcast messages to multiple nodes in an authenticated way, which can be achieved only via asymmetric schemes.

The use of asymmetric schemes in sensor networks has been usually considered as "nearly impossible" because they are power consuming and require a large amount of computational and storage resources. However, such schemes are very attractive, because they can ensure a higher degree of security while guaranteeing a greater flexibility and manageability than symmetric ones: thanks to them, any two sensors can establish a secure channel between themselves; moreover, as nodes do not share the same common key for encrypting/decrypting messages, the "capture" of some sensor devices will not affect the security of others. Rivest-Shamir-Adelman (RSA) algorithm [35] and Elliptic Curve Cryptography (ECC) [5] are among the most well known public key algorithms used in security systems, the latter being an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields.

Many protocols have been developed based on the ECC operations, as the Elliptic Curve Diffie-Hellman (ECDH) key agreement technique [5], which provides two communicating nodes with the possibility of achieving the same secret key without physically exchanging it across the network, and the Elliptic Curve Digital Signature Algorithm (ECDSA) [5] a variant of the Digital Signature Algorithm (DSA) that operates on elliptic curve groups, which can be used for signature generation and verification.

We analyzed some of the most recent solutions proposed in the literature to protect data exchanged among sensor nodes, and selected two available libraries, namely WMECC [6] and TinyPairing[7], both based on ECC, providing different levels of security and having different resource utilization profiles. We adopted these two libraries to build two different cryptosystems for securing data exchanged among nodes running a reference monitoring application (details are given in the next Chapter). The two developed cryptosystems were made available to the reconfiguration framework to build a reconfigurable security layer.

The next Chapter gives an overview of the chosen libraries, and presents and discusses the analysis conducted on their performance, energy consumption and intrinsic security level.

Chapter 4

Building a reconfigurable security layer for WSNs

This chapter describes two different possible configurations of the security layer of a WSN, based on the WM-ECC and TinyPairing libraries respectively. Such libraries have been selected among the most recent solutions based on ECC and were adopted to secure data exchanged among nodes running a reference monitoring application. The chapter presents and discusses the analysis conducted on performance, consumption and intrinsic security of these two cryptosystems.

4.1 The adopted security layer configurations

In this section we will give some details about the two cryptosystems (WM-ECC and TinyPairing) adopted to secure our reference monitoring application and used as the available configurations at the security layer in our reconfiguration strategy.

4.1.1 The WM-ECC-based cryptosystem

The first considered cryptosystem is based on the WM-ECC library [6], a publicly available open source implementation of a 160-bit ECC cryptosystem targeted to

MICAz, TelosB and Tmote Sky platforms, based on recommended 160-bit SECG elliptic curve parameters [secp160r].

Fundamental ECC operations are based on large integer arithmetic operations over finite fields as multiplication, division and modular reduction; in order to improve the performances of encrypting/decrypting operations, authors of WM-ECC library have exploited several optimizations and implemented parts of such operations directly in assembly language, in order to have a complete control of the register utilization.

WM-ECC provides support for all the ECC operations and gives an optimized implementation of the ECDSA protocol for digital signature generation and verification, relying upon techniques such as sliding-window and Shamir trick; it does not provide an implementation of the ECDH protocol, which we supplied by exploiting the basic ECC primitives and the main TinyOS components.

WM-ECC has been proved to be more computationally efficient than its major counterparts like TinyECC and EccM2.0: for example, on MICAz platform, TinyECC is 42% slower in signature generation, and on TelosB platform, the performance gap increases to 180%.

From an implementation point of view, WM-ECC is composed of 3 modules:

- Bint - provides optimized subroutines for large integer operations;
- ECC - based on the Bint module, implements all ECC operations;
- ECDSA - provides digital signature generation and verification primitives.

The WM-ECC library has been used to:

- implement the ECDH protocol for achieving a common secret key to be used for establishing a secure communication channel between the master and each of the motes;
- provide digital signature generation at the master side and verification at the mote side, by using the ECDSA primitives;

The secret shared key achieved via the ECDH protocol is used as a symmetric key for encrypting and decrypting the messages exchanged between the master (base station) and the motes with the Skipjack cypher [36], characterized by 80 bit keys and 64 bit blocks. The cypher has been used in such a way that the motes encrypt the result messages' payload after verifying the signature sent by the master, and the master decrypts such messages in order to get the results and forward them via the UART interface to the overlying application.

Let's first consider the implementation of the key agreement protocol. In order to implement the ECDH protocol the EcdhC component has been added to the WM-ECC library: this component uses the key_agree function implemented in the EcdhM module and accessible via the Ecdh interface; it is connected to the EccC component, providing all the ECC operations, and to the SHA1M_ncsu module, which provides the implementation of SHA-1 used by the Key derivation function (KDC).

The main steps performed by the protocol are the following:

1. after an initialization phase, the master generates a random key KM in the [1,n-1] interval, and performs a scalar product to calculate its own public point Q1=KM*P, where P is the base point on the curve.
2. at the same time the mote, after an initialization phase, generates a random key Km in the [1,n-1] interval, and calculate its own public point Q2=Km*P.
3. the master inserts Q1 into a message and sends such message to the mote.
4. the mote receives the message from the master, extracts Q1 and performs a scalar product in order to obtain the secret S=km*Q1; then it sends its public point to the master in a message.
5. the master receives the message from the mote, extracts Q2 and calculates the shared secret S=KM*Q2.

As for digital signature operations, we used the ECDSA primitives in order that:

1. the master node, at the arrival of a query from the UART interface, constructs a query packet with the received parameters, digitally signs it and then broadcasts it to the motes via the radio channel;

2. when receiving a query packet, a mote first verifies the digital signature and, if it turns to be successful, starts to sample the required physical values according to the query parameters; the retrieved results are collected and inserted into the payload of the response packet, which is previously encrypted before being sent back to the master with the secret key obtained after the ECDH protocol.

3. the master receives the response packet, extract its payload and decrypts it by using the shared secret obtained after the ECDH protocol. Then, the master returns the result values to the querying wrapper through the UART interface.

The encryption/decryption operations are carried out by means of the Skipjack cypher, realized by a custom TinyOS component implementing the BlockCypher TinyOS interface; such cypher uses the key derived by the ECDH protocol as the cryptographic key for encrypting the communication channel between the master and the motes. In order to reduce the overhead resulting by the cryptographic operations, we have chosen to encrypt only response packets sent by the motes to the master, and let only the master do decrypting operations to get the requested data.

We developed two different applications, respectively for the master and the mote side. The master application has been implemented so that it starts the ECDH protocol (step 1) when a timer expires: at the system setup a timer starts to run, and after 5 seconds an event is generated, handled by the master application which will calculate the master's public point and send it to the motes. The master application has been configured in order to digitally sign outgoing query packets addressed to the motes and decrypt the incoming response packets before sending the results to the wrapper. The mote application in turn, has been configured in order to be able to perform the ECDH protocol initiated by the master, verify the

digital signature of the incoming query packets, and encrypt all outgoing response packets.

In Figure 4.1 the execution of a query in a network, secured with the presented WM-ECC-based cryptosystem, is shown: at the system startup (red box in figure) the master node starts the ECDH protocol in order to achieve a common shared secret with each of the motes, then digitally signs every outgoing query packet and decrypts the incoming response packets before sending the results to the high level querying application. The mote in turn, is able to verify the digital signature of the incoming query packets, and to encrypt all outgoing response packets.

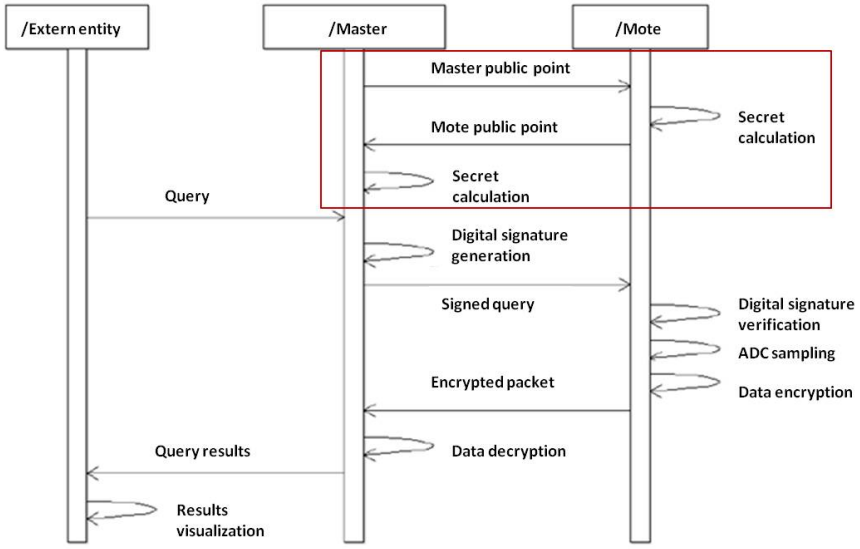

FIGURE 4.1: Query execution in a network secured with the WM-ECC-based cryptosystem

4.1.2 The TinyPairing-based cryptosystem

TinyPairing [7] is an open-source pairing-based cryptographic library for wireless sensors, designed to reduce memory occupancy (both ROM and RAM). It provides efficient and lightweight implementation of bilinear pairing, pairing-related functions and associated elliptic curve arithmetic operations such as scalar multiplication, point addition and more, and is the most efficient pairing based NesC implementation currently available.

In their implementation, the authors include three well-known pairing-based cryptographic schemes which have been employed in some recent solutions to secure WSNs: Boneh-Franklin Identity-Based Encryption (BF IBE) basic scheme [37], Boneh, Lynn and Shacham's Short Signature (BLS SS) scheme [38], and Boneh and Boyen's Short Signature (BB SS) scheme [39].

The entire library is written in nesC for TinyOS v2.x without using any hardware-dependent code, so it is easy to port to most of sensor platforms. Figure 4.2 shows the sequence of operations needed for executing a query in the TinyPairing-based cryptosystem that we set up.

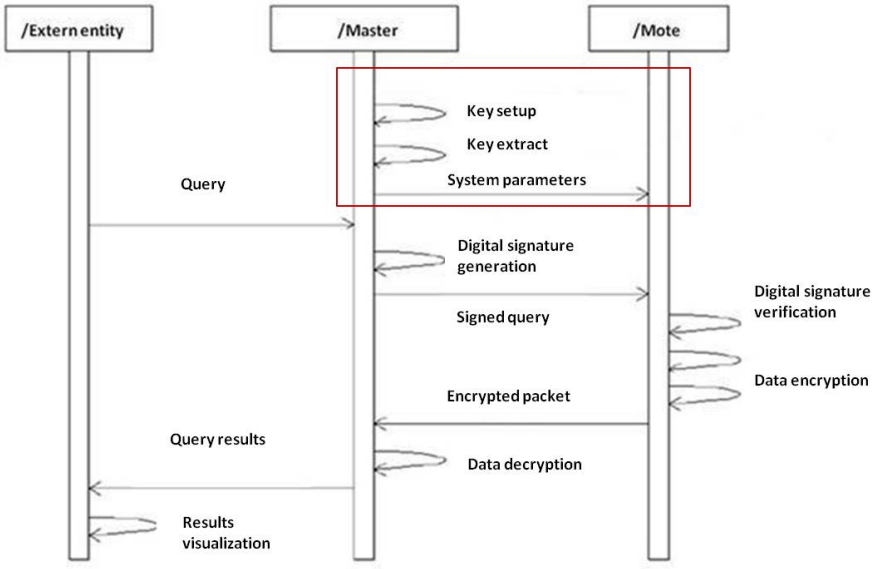

FIGURE 4.2: Query execution in a network secured with the TinyPairing based cryptosystem

- In the pre-deployment phase (red box in figure), carried out in a protected environment, the master node is assigned a unique 8 byte identifier based on manufacturer and serial number information and performs two operations: in the *key setup* operation it uses a randomly chosen large integer (secret master key) and generator point to obtain a *public point*; in the *key extract* operation, the master key and the unique 8 byte ID are used to achieve the *master private key* (dID) associated with its public ID, which will be used in decrypting operations.

The master ID, its public point and the random generator point are sent to each of the motes, as this information is necessary in signature verifying and response encrypting operations.

- At the arrival of a query from the UART interface, the master builds a query packet with the received parameters, digitally signs it with its master key (according to the BLS SS scheme) and then broadcasts it to the motes via the radio channel;
- When receiving a query packet, a mote first verifies the digital signature using the master public point and generator point, achieved during the pre-deployment phase and, if the verification turns to be successful, starts to sample the required physical values according to the query parameters; each retrieved sample is collected and inserted into the payload of a response packet, which is encrypted using the master *ID* (according to the BF IBE scheme) before being sent back to the master;
- The master receives the response packet, extract its payload and decrypts it with its private key *dID*; then, the master returns the result values to the high level querying application through the UART interface.

As illustrated, the query process is very similar to the WM-ECC based protocol, except for the key agreement phase that is more secure: only the master node calculates key information and sends the necessary parameters to the motes in order to let them perform cryptographic operations.

4.2 Security and Performance Analysis

The introduction of security mechanisms within a sensor network is a desirable feature but it may introduce a very heavy overhead that must be addressed by any sensor networks developer and deployer.

We conducted specific analysis aimed at:

- measuring and comparing performance and power consumption of the two proposed cryptosystems
- defining the intrinsic level of security provided by each cryptosystem, based on the security requirements they are able to ensure (static security analysis)

These information, in fact, should be taken into account by any designer to meet her security and performance requirements.

For our experiments we adopted two different hardware platforms, commonly used by the scientific community, and both provided by CrossBow:

- **TelosB** motes, provided with a MSP430, 16 bit-processor, a CC2420 radio chip, 10kB internal RAM, 48kB flash programming memory and 1024k memory for measurements,
- **MicaZ** motes, based on a MPR2400, 8 bit-processor, a CC2420 radio chip, 4k byte RAM, 128k flash programming memory and 512k memory for measurements.

The following sections discuss the experimental results obtained from performance, power consumption and security level evaluation for the proposed cryptosystems.

4.2.1 Performance evaluation

In order to evaluate the performance of the two given cryptosystems, we measured the following parameters on both hardware platform introduced in the previous section:

- latency introduced in the whole monitoring architecture
- overhead introduced by the protocol
- resource occupation to elaborate cryptographic functions on single nodes

Details about the achieved results are given in the following.

4.2.1.1 Latency

Cryptosystem	**WM-ECC**		**TinyPairing**	
	TELOSB	MICAZ	TELOSB	MICAZ
Initialization	1,428436	1,258555	17,083984	10,180664
Key Exchanging	1,583069	0,062666		
Sign	1,49704	1,344894	8,694336	5,540033
Verify	2,24049	1,984192	30,163086	17,867188
Data Encrypting	0,001221	0,00034	27,884766	15,972656
Data Decrypting	0,001252	0,000334	13,019531	7,838867

TABLE 4.1: *Latency in seconds*

In Table 4.1 the latency introduced by the security protocols is reported and compared. As illustrated, WM-ECC presents much better performance than TinyPairing, thanks to the adoption of a hybrid approach relying on a fast symmetric cypher, compared to the asymmetric scheme adopted in TinyPairing. As expected, increasing the security of the protocols is paid against a performance loss. From a network designer point of view, an encryption time of abut 28 seconds (TinyPairing) implies that we cannot choose a sample period shorter than this interval, otherwise, without buffering capabilities, we will loose samples. Clearly, this could have a high impact on the monitoring application functional requirements.

By observing the table, it can be easily noted that signature and verification as well as encryption and decryption operations are strongly asymmetric in computation time for the TinyPairing cryptosystem. This is due to the specific sequence of additional operations that are needed to implement the TinyPairing mechanism.

As for the signature verification, in fact, it requires a double call of the same pairing function with different input parameters; as for the encryption, it requires a sequence of different operations to initialize the ECC-based algorithm (hash-to-point, point scalar multiplication, ...) as illustrated in [7].

Latency could be reduced by buffering some samples and encrypting them all together (at most 4 samples in this case, as the maximum data block that can be encrypted in a single step is 8 bytes and a single sample only occupies 2 bytes).

4.2.1.2 Packet overhead

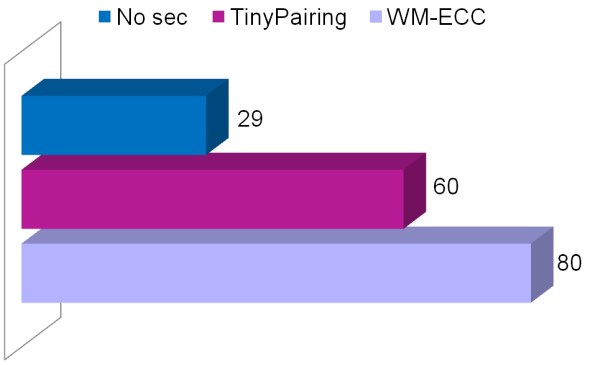

FIGURE 4.3: Packet length in *bytes*

In Figure4.3 the packet overhead introduced by the two cryptosystems is compared with the not secure version. In order to make the security application feasible, we had to increase the payload length in both cases from the default 29 bytes.

In particular, for the WM-ECC based network, we sized the packet to 80 bytes to enable the transmission of the public points and digital signatures; for the TinyPairing based network, we sized the packet to 60 bytes for the transmission of the encrypted packets. In general, packet size is a very crucial parameter as this also has a bad impact on battery consumption, usually very high during the transmission phase; it could be reduced by performing a little variation in the protocol consisting in splitting packets in two or three portions in order to fit smaller dimensions.

4.2.1.3 Memory occupancy

As for resource occupancy, encryption and authentication operations produce an additional cost in terms of memory and CPU. In Figures 4.4 and 4.5 the memory overhead is reported in terms of RAM and ROM usage respectively on master and end-nodes.

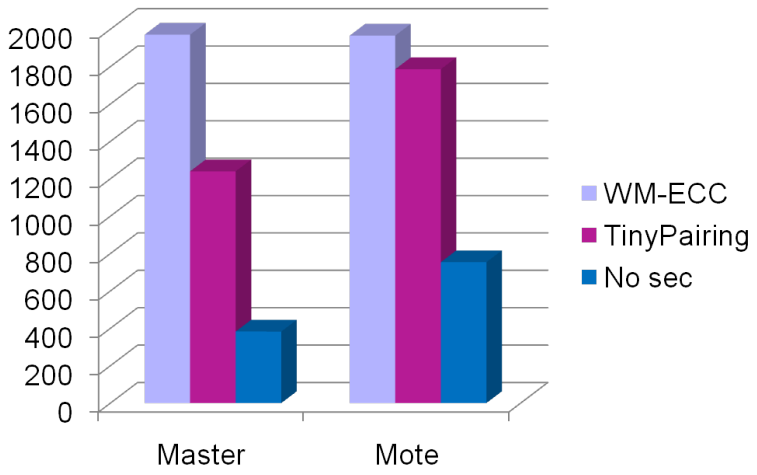

FIGURE 4.4: RAM occupancy in *bytes* for master and mote applications

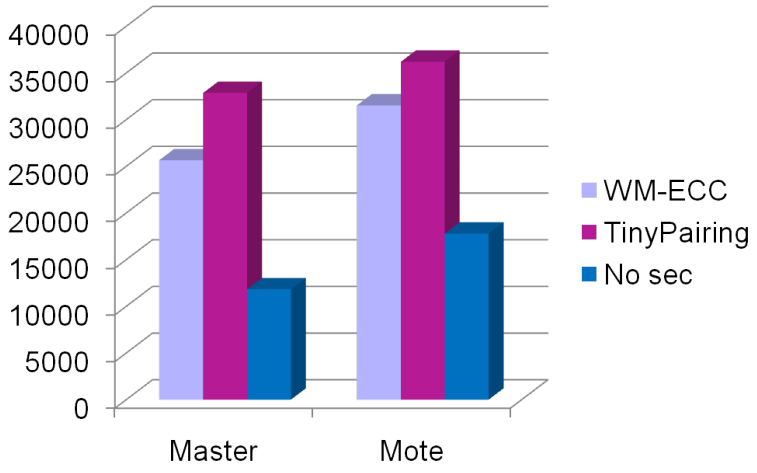

FIGURE 4.5: ROM occupancy in *bytes* for master and mote applications

As illustrated, the secure query applications imply a higher RAM and ROM usage both on master and mote sides (even 5 times more), compared to the simple query application without security. Anyway, this is not a problem as Telosb motes have 48K bytes of memory and other common platforms, as MicaZ, have even more storage capabilities (128K).

As final remark, from a design point of view, the resulting values can be considered acceptable assuming to deploy sensor nodes that only run a monitoring application at a time, and depending on the available resources and on the security requirements, many solutions can be adopted based on different cryptographic schemes and protocols.

4.2.2 Power consumption evaluation

In order to measure the power consumption associated with the two cryptosystems, we adopted AVRORA [40], a set of simulation and analysis tools for programs written for the AVR microcontroller produced by Atmel (installed on the Mica2 sensor platforms).

We considered separately the three phases of initialization, signing and sampling, and measured the energy consumption associated to CPU and radio transceiver for these phases. Results are shown in Table4.2.

Cryptosystem	**WM-ECC**		**TinyPairing**	
	CPU	RADIO	CPU	RADIO
Initialization	2,48423E-09	5,64598E-09	2,27555E-10	1,04346E-09
Sign	1,92191E-09	6,95544E-09	1,25765E-09	4,67708E-09
Sampling	1,92203E-09	6,9554E-09	2,135E-09	7,5457E-09

TABLE 4.2: *Power consumption in Joules*

As expected, the cryptosystem based on TinyPairing is characterized by an higher energy consumption, especially in the sampling phase, that is the most critical one as it is performed periodically and constitutes the main operation executed by sensor nodes.

4.2.3 Security evaluation

From the security point of view, the cryptosystem based on WM-ECC does not authenticate the public keys, thus allowing man-in-the-middle attacks in the key exchange phase. Moreover, sensitive data are encrypted with a symmetric cipher, that increases vulnerability of the network.

TinyPairing instead, adopts an asymmetric scheme and is much more secure in the initialization phase as it does not use a key exchange mechanism. As stated in Chapter 2, the intrinsic level of security of the two configurations can be represented by an attack coverage table, identifying, for each configuration, what attacks it is able to thwart or, dually, what requirements it is able to satisfy.

Conf	*Man-in-the-middle*	*Eavesdropping*	*Brute force*	*Replay attack*
WM-ECC	non-auth key	yes	weak symm	no
TinyPairing	auth key	yes	asymm	no

TABLE 4.3: Attacks Coverage Table for the considered cryptosystems

In table 4.3, coverage is not defined as an ON/OFF property, but it is defined through a score that corresponds to the level of protection provided by the configuration with respect to each considered attack.

Chapter 5

Enforcing WSN reconfiguration: implementation details and evaluation

This chapter first discusses some existing approaches to WSN reconfiguration and then proposes two reconfiguration mechanisms, designed to perform security layer and physical layer reconfiguration respectively. Finally, it discusses the theoretical and experimental analyses conducted to prove that the proposed reconfiguration mechanisms are effective in increasing the complexity for the attacker and, consequently, in decreasing the probability of completing a successful attack.

5.1 WSN reconfiguration in literature

Several reconfiguration mechanisms have been proposed for WSNs, mainly based on network reprogramming. Embedded systems reprogramming has been widely addressed in the literature, as it is fundamental to perform management and maintenance tasks – e.g., software updates, bug fixes, parameter tuning – in those applications where it is not possible or convenient to physically access and manually reconfigure deployed nodes.

In [41], authors provide an interesting review of reprogramming frameworks, design challenges, existing systems and approaches, and evaluation metrics for WSN. They highlight the importance of providing algorithms and protocols that take into account the limited resources of sensor nodes, the typical unreliability due to wireless channels and dynamic topology and the scalability problems, as well as the limitations of the most common operating system running on sensor nodes, TinyOS: it does not provide reliable memory allocation mechanisms or multi-threading models, and generates monolithic compiled programs so that components cannot be separated and reprogrammed independently.

Several existing approaches for sensor network reprogramming perform a *full-image replacement*, consisting of completely updating the image of the application running on nodes.

Deluge [42] is a reliable data dissemination protocol for propagating large data objects (larger then the RAM size of a node) from one or more source nodes to many other nodes over a multi-hop network. The protocol operates as a state machine where each node can be in one of three states at any time: MAINTAIN, RX, or TX. Deluge implements a three phase Advertise-Request-Data gossip protocol, in which data is only pushed by a sender upon receiving an explicit request for data from an immediate neighbor. A node in the MAINTAIN state is responsible for ensuring that all nodes within communication range have the newest version of the object profile and all available data for the newest version and this is obtained by occasionally advertising a summary representing the current version of its object profile and the set of pages (transfer units) from the object which are available for transmission. When receiving an advertisement for a needed page, a node moves to the RX state and is responsible for actively requesting the remaining packets required to complete page; after a node makes a request, it waits for a response and makes subsequent requests if some of the data was lost in the process by using a negative ack mechanism. Deluge limits a node to λ requests before returning to MAINTAIN; however, if progress (measured as reception rate of data) is above some threshold Deluge allows the node to continue making requests. While being in the MANTAIN state, if a node receives a request for a

given page locally available, it makes a transition to the TX state, in which it is responsible for broadcasting all requested packets for the page (continuing to service any subsequent requests for data from the same page) until all requested packets have been broadcast; then it goes back to MAINTAIN state. In [**?**] a secure version of Deluge is provided.

Differential image replacement consist of disseminating only changes between a deployed executable and a new image, reducing overhead. In [43], authors present an incremental hardware-independent network programming mechanism which reprograms one-hop wireless sensors quickly by transmitting the incremental changes for the new program version using the Rsync algorithm, which finds the shared code blocks between the two program images and allows to distribute just the key changes.

Zephyr [44] uses an optimized version of the Rsync algorithm to perform byte level comparison between the old and new executables and provides a multi-hop reprogramming protocol; moreover, before performing byte level comparison, Zephyr performs application level modifications on the old and new versions of the software to mitigate the effect of function shifts so that the similarity between the two versions of the software is increased and the delta size is decreased.

In [45] R^2 is proposed, being a unified approach to mitigate both effects of function shifts and data shifts by using relocatable code: this approach obtains a higher degree of similarity by keeping all references in the instructions the same in both program versions and it also makes efficient use of the memory while not degrading the program quality.

Different approaches are based on *dynamic Operating Systems*, aimed at obtaining a modular structure for compiled applications in order to be able to update only components that actually change.

TinyCubus [46] is based on TinyOS, which is primarily used as a hardware abstraction layer. For TinyOS, TinyCubus is the only application running in the system, and all other applications register their requirements and components

with it. TinyCubus provides a set of management components that make it possible for several implementations of an application to coexist on a node, and it is responsible for dynamically loading components into the sensor's memory on an as-needed basis.

Finally, Dynamic TinyOS [47] alters the compilation process of TinyOS in order to generate an executable consisting of multiple, replaceable objects, so that, after deployment, updates can be disseminated to replace existing objects on sensor nodes.

All these replacement techniques are based on dissemination of application binary images over the network, thus arising security concerns and a substantial overhead in terms of delays and power consumption and, at the state of the art, they are not very suitable for the MTD approach.

5.2 Our approach to reconfiguring WSNs

In this section we provide two reconfiguration mechanisms for security layer and physical layer respectively. To this aim, we will explicitly refer to a WSN architecture, but the discussion can be easily generalized to any distributed architecture having analogous constraints.

As previously highlighted, security is a fundamental concern in WSNs, as they are widely adopted in several critical application domains; nevertheless, because of their peculiar features – constrained processing and storage capabilities, limited energy resources, highly dynamic topology and mobility, frequent and failures, – providing security is not a straightforward task. The introduction of security mechanisms has a strong impact on performance and resource consumption, that often represent a limiting factor. For this reason, the adoption of a complex cryptosystem (i.e. based on public key primitives) for the entire network life could be desirable by a security point of view but not recommendable or not feasible in practice.

The proposed reconfiguration approach is able to overcome these concerns, as it allows to keep an acceptable level of security in the network, by leveraging not only the intrinsic features of the running cryptosystem, but also other features, such as the physical configuration and the application interfaces, other than the reconfiguration mechanism itself, as previously shown. This way, the use of simpler cryptosystems for a short period of time can be preferred to the adoption of a single strong but high-consuming cryptosystem. Indeed, a more secure and complex cryptosystem is still worthy of implementation and use, to cope with new security requirements for instance, or to thwart specific detected attacks.

Referring to TinyOS, the most commonly adopted operating system for WSNs, security mechanisms could be implemented either as independent TinyOS components or as different static libraries wired in the same component, whose functions are called by applications to ensure security requirements. Reconfiguration of the security layer and of the application interfaces could be easily achieved by including the implementation of all the available solutions into the firmware installed on the device, and activating the desired configuration through software switches and proper protocols. Firmware reconfiguration instead, can be performed by adopting node reprogramming techniques, that will be shown in details later.

Figure 5.1 shows a sequence diagram representing the network operation in presence of the reconfiguration mechanism. In the INIT phase each node is loaded with the application image: according to the above discussed strategies, this could be either an enriched application featuring different cryptosystems, or a collection of different application images, each implementing a different cryptosystem and allocated in the node's flash memory. Each of the available cryptosystems must be initialized in order to be used; this is also performed in the INIT phase, which usually is carried out in a protected environment.

At each time, only one of these cryptosystems will be considered *valid* for a given node and used for actual communication. The time interval between two subsequent updates is referred to as the *validity interval*.

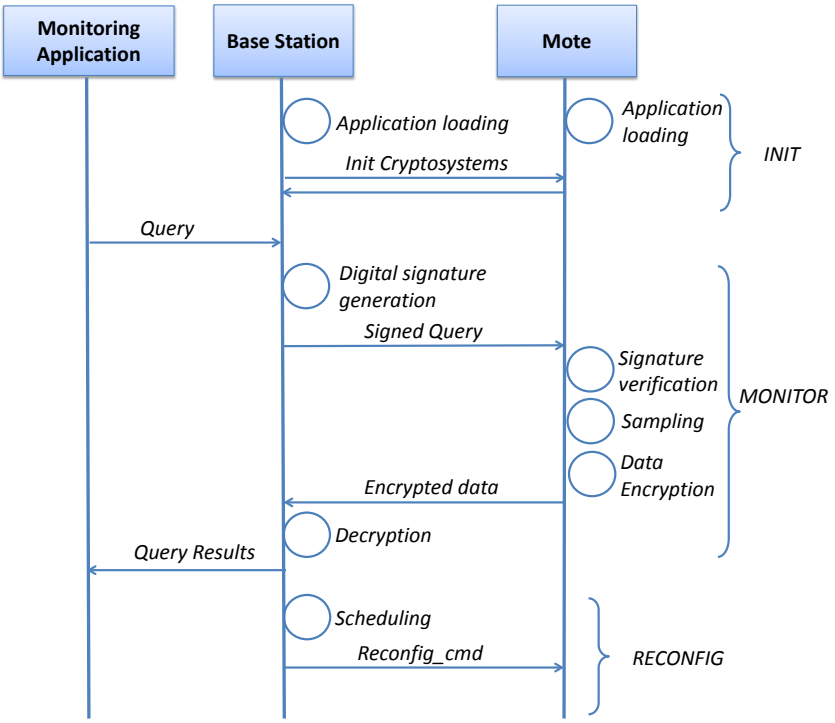

FIGURE 5.1: Reconfiguration sequence

As illustrated in the MONITOR phase of Figure 5.1, all queries sent by the base station in a validity interval will be signed using the valid cryptosystem's signature protocol, and the signature will be verified by motes adopting the current valid cryptosystem associated with the base station. Similarly, all responses sent by a mote in a validity interval will be encrypted using its valid cryptosystem's encryption method, and decrypted by the base station using the decryption method belonging to the cryptosystem currently associated with that node.

As illustrated in the RECONFIG phase of Figure 5.1, the update process executed by a node consists in switching to a selected image or to another cryptosystem from the available pool (either after receiving a command or making its own decision). Required cryptographic operations on future outgoing messages will be performed using the newly selected cryptosystem.

The selection of a new configuration is performed by a security-driven scheduler, aiming at increasing the level of security which can be measured in terms of the probability of successfully completing an attack, as shown later. In particular, the

scheduler will define both the reconfiguration frequency and the new configurations.

In the following subsections, we will focus on the security layer and firmware reconfiguration respectively, presenting two innovative approaches to provide reconfiguration, with some implementation details.

5.2.1 Security Layer Reconfiguration

In our reference monitoring application, queries are signed by the base station for authentication purposes, and reply messages are encrypted for ensuring confidentiality and integrity, as shown in the previous Chapter with respect to the adopted cryptosystems. The security layer performing these operations can be designed to implement different cryptographic protocols, depending on the required security level and on available resources. As previously said, the basic idea of the proposed approach is to dynamically change the security layer, by switching between two or more different available implementations.

As said in Section 2.2.4, both the base station and the motes can trigger an update in response to an event, such as detection of malicious activity or timer expirations. In the simplest scenario, nodes could decide to perform an update by selecting, based on a shared strategy, the next valid cryptosystem once a shared timer expires; this approach does not require to exchange information about the adopted cryptosystems, but it relies upon strict synchronization, which itself can not be achieved without adding considerable overhead.

In order to control the overhead and increase network flexibility and diversity, we introduce a different reconfiguration protocol: each node can decide independently when to update, and an identifier of the cryptosystem used to encrypt a message is coded in the message itself, so that each receiving node, sharing the same security configuration, is able to properly handle it.

As illustrated in Figure5.2, the cryptosystems' parameters (i.e. the cryptographic keys) could be either pre-loaded on all network nodes in the INIT phase, or dynamically determined in each RECONFIG phases according to the available key agreement mechanism.

In the first solution all the cryptographic keys are stored in each node for the whole lifetime of the network, and they can be used as master keys for generating new keys.

In the second solution, instead, each time a node triggers a local update, it has to start an initialization procedure according to the chosen cryptosystem to establish the cryptographic parameters. The initialization procedure usually involves exchanging messages between the base station and each of the motes, thus exposing the procedure itself to attacks (such as the man-in-the-middle attack for key agreement operations) and introducing delay in delivering response packets from the motes' point of view.

In Figure 5.2, a typical scenario of security protocol reconfiguration is illustrated. Assume that each node is provided with a pool of different cryptosystem implementations, which are identified by a unique ID. In the INIT phase, the base station and the motes agree on cryptosystems' parameters; initialization depends on the specific adopted cryptosystem (they can be public points for an ECC based cryptosystem, or system parameters for an identity based [48]) and can be performed in a secure environment in the pre-deployment phase or later.

As shown in Figure 5.2, after initializing the N available cryptosystems, each node can independently choose the valid cryptosystem to adopt in order to perform cryptographic operations in the current validity interval. In particular, the base station choses the cryptosystem it will use to digitally sign the outgoing queries in order to assure authentication (CRYPTO(i) in figure); any mote receiving the query message will use the cryptosystem whose ID is included in the message itself to verify the signature. Similarly, after verifying the signature, any mote encrypts data according to the local selected cryptosystem (CRYPTO(j) and CRYPTO(w)

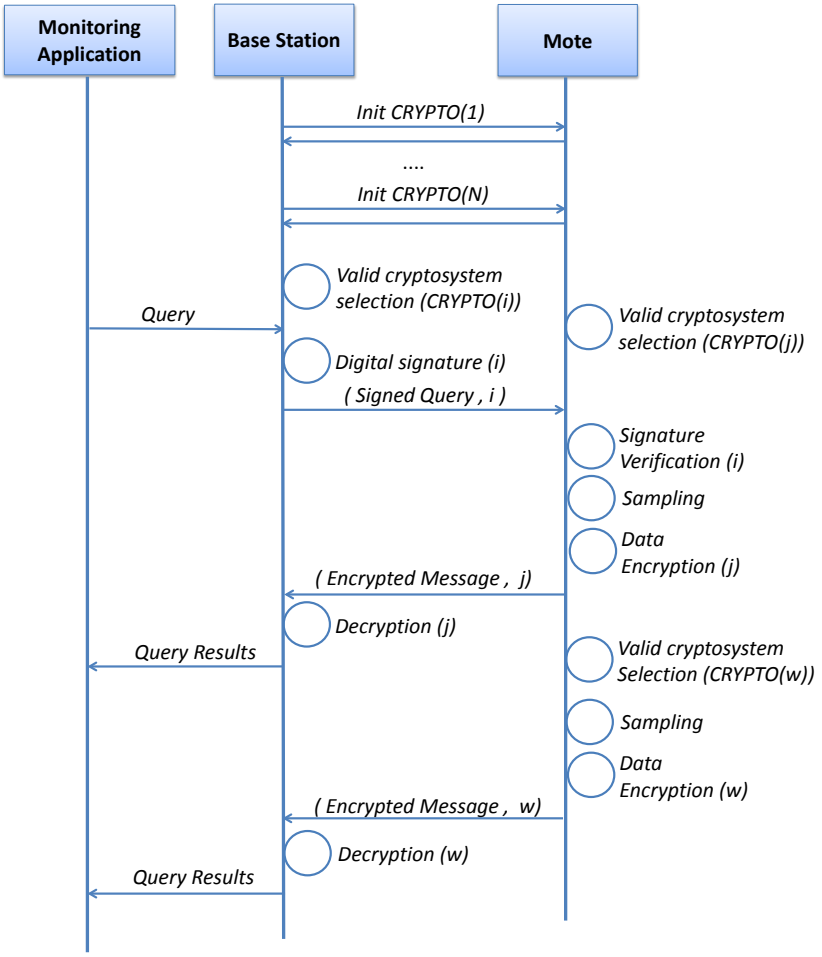

FIGURE 5.2: Security protocol reconfiguration

in figure), and the base station will use the ID included in the reply messages to decrypt them.

The main advantages of this solution relate to improved overall performance. In fact, there is no latency to swap from a cryptosystem to another and we do not need to stop the monitoring application during the reconfiguration; even battery consumption is not affected by this solution.

As for the security of this strategy, if an attacker is aware of the packet format, she could try to manipulate some fields of the packet, such as those coding the cryptosystem ID and its parameters, so that nodes are no longer able to communicate. As for query messages, their payload is signed with the base station's private key, so that if any field is altered during transmission the signature verification at mote's side will not succeed. This could lead to a denial of service attack, as

motes will not be able to verify the authenticity of queries and will not perform the required actions. To detect this attack, a timeout could be set by the base station each time a query is sent; if no results come before the timeout expires, the query could be sent again to cope with possible message losses; if no results are returned after few attempts, an alert could be raised. The cryptosystem ID and information about its parameters are coded in the response messages, too, as they are necessary to decrypt the message. An attacker could alter such fields as the messages are not authenticated, but the base station will not be able to decrypt them causing the loss of some response messages (denial of service). As a typical network is composed of many redundant mote nodes, this situation can be considered as not critical.

5.2.2 Physical Layer Reconfiguration

As mentioned in Section 5.1, the Deluge T2 Network reprogramming framework enables the reconfiguration of a node by sending via radio the new binary image that should be loaded on the node. As Dutta et al. pointed out in [49], this approach is unsafe and too battery consuming. We implemented a different approach to remotely reconfigure each node in the network; we decoupled the reconfiguration mechanisms from the components to enforce the new configuration according to a scheduling policy.

To this aim, we designed a reconfiguration application by augmenting several components of the Deluge framework. In particular, we implemented new reconfiguration functionalities to enable a single node to swap to a new image that was previously pre-loaded on its storage. The reconfiguration application is defined by wiring new components specifically designed to manage external reconfiguration commands, and components designed to manage the images loaded on the node storage.

As illustrated in the right part of Figure 5.3, the proposed reconfiguration application consists of three main components:

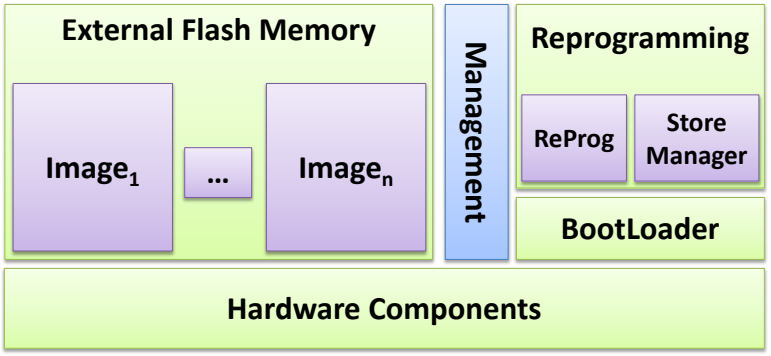

FIGURE 5.3: Reconfiguration Application components

1. a bootloader component,
2. a reprogramming component,
3. a management component.

The *bootloader component* is a persistent layer in the architecture, which can enforce the chosen reconfiguration mechanisms. This component is intended for TinyOS and it provides needed functionalities to program the node with an already stored program image. The parameters passed to this component are specified in the external command and indicate the location of the binary in the external flash memory to program the node's microcontroller. When reprogramming is requested, the bootloader will erase the program flash and write the new binary to it. On completion, it jumps to the first instruction of the new application.

The *reprogramming component* is the core of the *reconfiguration application*; in this implementation it accepts commands from the base station, but can be extended to implement a de-centralized reconfiguration approach. This component is built by connecting two primary subcomponents: the *ReProg* and the *Storage-Manager*. The ReProg component is an extension of the NetProg component of Deluge T2. It handles a reprogramming request from the network by providing a dedicated API to initialize a reconfiguration process. When a node wants to perform a reconfiguration, it only has to invoke this API by specifying the name of the new binary in the flash memory to load. Subsequently, the ReProg sets the environment variables needed by the bootloader component and reboots the

node. The StorageManager component deals with image name resolution, mapping names of program images to their respective physical addresses in the external flash memory.

The *management component* has a master (base station) and mote side; it is used to initialize the mote and deploy different images. Usually this operation is done in a secure environment and it is accessible only during the initialization. The master-side *management component* has been derived from the *tos-deluge* application of the Deluge T2 Framework, and it is called mote-manager. This component allows to inject one or more images into the mote by writing directly into nodes' external flash memory volumes. It is also possible to erase a volume and ping the status of a mote to get information about already injected images.

Finally, the reconfiguration application runs on a workstation connected to the base station, which implements the reprogramming scheduler. As discussed in the next section, the reprogramming frequency and the new configuration to load can be chosen to balance overhead and attack probability.

The proposed solution introduces considerable advantages in terms of security and overall performance with respect to WSN reprogramming approaches based on code dissemination. In fact, the reconfiguration time is now not dependent on the image size and the network topology as the images are not sent over the network but pre-loaded via a serial interface. Furthermore, this approach avoids any security risk in the dissemination and reduces the battery consumption as the messages sent are just commands to swap from an image to another one.

The swapping latency is considerably reduced, too; we experimented a reduction of one order magnitude with respect to the original Deluge approach: from 50 seconds to send a 40Kb image implementing a monitoring application secured with WM-ECC, to about 6 seconds to perform the swap. The only drawback is that we need to stop the monitoring application and any query being executed in order to swap to another cryptosystem. However, any approach based on full image replacement presents a similar issue. Furthermore, due to storage limitation, we can only pre-load a limited number of images on board.

Let us consider attacks aimed at undermining this reconfiguration mechanism. An attacker could perform a replay attack on control packets sent by the base station and containing a reconfiguration command, in order to control communication or just perform a denial of service attack by forcing the motes to continuously swapping images. Again, this could be avoided by introducing a sequence number for reconfiguration commands.

5.3 Theoretical Evaluation

The objective of our approach to WSN security is to increase the complexity for attackers and impair their ability to successfully discover cryptographic keys or complete various other types of attacks. In the following we show theoretically how the proposed approach decreases the probability of an attacker successfully discovering cryptographic keys by brute force attacks.

We assume that the attacker sequentially tests all possible keys for a given cryptosystem. In the worst case for the defender, the attacker can recognize that the cryptographic system has changed and restart the attack. The attacker may not know what specific cryptosystem is being used in each interval. In this case, he will try all cryptosystems from a set of possible candidates. Clearly, more sophisticated types of brute force attacks exist today, compared to the one described here. Such types of attacks leverage specific properties of a given cryptographic system. However, our goal here is to evaluate the benefits of the proposed mechanism with respect to the case where the cryptographic system is never changed during the lifetime of the wireless network. Therefore, the specific method of attack is not relevant in our analysis.

Given a time interval $[t_i, t_j]$, we use $\Pr(success([t_i, t_j]))$ to denote the probability that the attacker will successfully discover the key between t_i and t_j when a single cryptographic system is used during the interval $[t_i, t_j]$. Similarly, we use $\Pr(success([t_i, t_j], n))$ to denote the probability that the attacker will successfully discover the key between t_i and t_j when the interval $[t_i, t_j]$ is broken down into

n validity intervals and a different cryptosystem is used in each such intervals. Clearly, $\Pr(success([t_i, t_j], 1)) = \Pr(success([t_i, t_j]))$. We can prove the following theorem.

Theorem 5.1. *Let $[0, T]$ be an observation interval, and let $n \in \mathbb{N}$ be and integer greater than or equal to 2. Then the following inequality holds.*

$$Pr(success([0, T], n)) \leq Pr(success([0, T])) \tag{5.1}$$

Proof. The probability that the attacker will successfully break the cryptosystem between 0 and T when the interval $[0, T]$ is broken down into n validity intervals – and a different cryptosystem is used in each such intervals – can be written as

$$\Pr(success([0, T], n)) = 1 - \Pr(\neg success([0, T], n)) \tag{5.2}$$

The probability $\Pr(\neg success([0, T], n))$ that the attacker does not succeed by time T is the probability that he does not succeed in any of the n validity intervals.

$$\begin{aligned} \Pr(\neg success([0, T], n)) = \Pr\left(\neg success\left(\left[0, \tfrac{1}{n} \cdot T\right]\right)\right. \\ \wedge \neg success\left(\left[\tfrac{1}{n} \cdot T, \tfrac{2}{n} \cdot T\right]\right) \\ \left. \wedge \ldots \wedge \neg success\left(\left[\tfrac{n-1}{n} \cdot T, T\right]\right)\right) \end{aligned} \tag{5.3}$$

The events $\neg success\left(\left[0, \frac{1}{n} \cdot T\right]\right)$, $\neg success\left(\left[\frac{1}{n} \cdot T, \frac{2}{n} \cdot T\right]\right)$, …, $\neg success\left(\left[\frac{n-1}{n} \cdot T, T\right]\right)$ are clearly independent, thus $\Pr(\neg success([0, T], n))$ can be computed as follows.

$$\begin{aligned} &\Pr(\neg success([0, T], n)) = \\ &\quad \textstyle\prod_{i=0}^{n-1} \left(1 - \Pr\left(success\left(\left[\frac{i}{n} \cdot T, \frac{i+1}{n} \cdot T\right]\right)\right)\right) \end{aligned} \tag{5.4}$$

As the probability that the attacker can break the system in a given interval is directly proportional to the length of the interval itself, we can conclude that, for

all $i \in [0, n-1]$, $\Pr\left(success\left(\left[\frac{i}{n} \cdot T, \frac{i+1}{n} \cdot T\right]\right)\right) = \frac{\Pr(success([0,T]))}{n}$. This conclusion relies on the simplifying assumption that the different cryptosystems used in our framework are equivalent in terms of attack time. Generalizing this result to the case of heterogeneous cryptosystems is straightforward, but it is omitted for reasons of space. Additionally, the above conclusion assumes that the interval $[0, T]$ is larger than the time needed to complete a full brute force attack[1]. Then, Equation 5.4 can be rewritten as follows.

$$\begin{aligned} \Pr(\neg success([0,T],n)) &= \prod_{i=0}^{n-1}\left(1 - \frac{\Pr(success([0,T]))}{n}\right) \\ &= \left(1 - \frac{\Pr(success([0,T]))}{n}\right)^n \end{aligned} \tag{5.5}$$

In order to complete the proof, we need the results of another theorem:

Theorem 5.2. *Let $x \in [0,1]$ be a real number and let $n \in \mathbb{N}$ be an integer number. The following inequality holds.*

$$\left(1 - \frac{x}{n}\right)^n \geq 1 - x \tag{5.6}$$

Proof. Using the binomial theorem, the expression $\left(1 - \frac{x}{n}\right)^n$ can be expanded as follows.

$$\left(1 - \frac{x}{n}\right)^n = \sum_{k=0}^{n} \binom{n}{k} \left(-\frac{x}{n}\right)^k = 1 - x + \sum_{k=2}^{n} \binom{n}{k} \left(-\frac{x}{n}\right)^k \tag{5.7}$$

To complete the proof, it is sufficient to show that the alternating series $\sum_{k=2}^{n} \binom{n}{k} \left(-\frac{x}{n}\right)^k$ is greater than or equal 0. As the first term in the series is positive, we only need to show that all the terms have decreasing absolute values. In order to do so, we now show that the ratio between two consecutive terms is greater than 1.

[1] If $\Pr(\neg success([0,T])) = 1$, then there may exist a sub-interval $[t_i, t_j]$ of $[0,T]$ such that $\Pr(\neg success([t_i, t_j])) = 1$.

$$\left| \frac{\binom{n}{k}\left(-\frac{x}{n}\right)^{k}}{\binom{n}{k+1}\left(-\frac{x}{n}\right)^{k+1}} \right| = \frac{\frac{n!}{k!\cdot(n-k)!}}{\frac{n!}{(k+1)!\cdot(n-k-1)!}\cdot\frac{x}{n}} = \frac{n\cdot(k+1)}{(n-k)\cdot x} \tag{5.8}$$

It is clear that the quantity at the right end side of Equation 5.8 is greater than 1, as $n\cdot(k+1)\geq n$ and $(n-k)\cdot x\leq n$. □

Using Theorem 5.2, we can conclude that

$$\left(1-\frac{\Pr\left(success\left([0,T]\right)\right)}{n}\right)^{n}\geq 1-\Pr\left(success\left([0,T]\right)\right) \tag{5.9}$$

Combining Equations 5.2, 5.5, and 5.9, we can write

$$1-\Pr\left(success\left([0,T],n\right)\right)\geq 1-\Pr\left(success\left([0,T]\right)\right) \tag{5.10}$$

Equation 5.1 follows directly from Equation 5.10. □

In conclusion, Theorem 5.1 shows that, in theory, the proposed mechanism is effective in reducing the probability that the attacker will successfully discover currently used cryptographic keys in a given amount of time. In other words, it will take more time for the attacker to break the system. Experiments reported in the next section confirm this result.

5.4 Simulation experiments

In this section, we show, through simulation experiments, that the proposed mechanisms are effective in increasing the uncertainty and complexity for the attacker, thus confirming the theoretical results illustrated in the previous section. We assume that the attacker's goal is to break the system by attempting to discover the cryptographic keys used to protect communication. Additionally, in this set of experiments, we assume that security reconfiguration is performed by randomly

switching among three different cryptosystems. In order to evaluate the effectiveness of security-driven reconfiguration – even in some adverse conditions – we assume that an attacker is able to understand when the adopted cryptosystem changes and what kind of cryptosystem is used at each time (e.g., by observing control messages sent over the network by the base station in the node reconfiguration strategy, or control flags present in data packets in the protocol reconfiguration strategy).

Many types of cryptographic attacks can be considered. In our case, an attacker can only observe encrypted packets traveling on the network, containing information about sensed data, and it can perform a brute force attack on captured packets by systematically testing every possible key for the current (known) valid cryptosystem – assuming that it is able to determine when the attack is successful. Given a finite key length and sufficient time, a brute force attack is always successful; depending on the adopted algorithm, some well-known weaknesses can be exploited (e.g., *weak keys* or *equivalent keys*) to run a more efficient type of attack. Nevertheless, for ease of presentation, we consider brute force attacks carried out by sequentially testing all possible keys.

In particular, we consider two possible cases:

1. the attacker knows the encryption algorithm and the key length associated with the algorithm, therefore he systematically tries all the possible keys of that length;
2. the attacker knows the encryption algorithm but does not know the key length associated with it, thus he systematically tries all the possible keys for a given set of key lengths.

Case 1 represents the *worst case* for the defendant, as the attacker has the deepest knowledge about the adopted cryptosystem. Case 2 will be referred to as the *intermediate case* in the following discussion. The *best case* for security is represented by an attacker who does not know anything about the adopted cryptosystems, and therefore tries all the possible keys for a given set of key lengths and a given

set of cryptosystems. In the following we do not discuss this case explicitly, as it can be considered as a generalization of the intermediate case.

The time to complete a brute force attack clearly depends on many factors, including the cryptographic algorithm, the key space, and the attacker's elaboration resources. Commonly adopted cryptosystems use large keys, and require a huge computational effort to be broken with a brute force attack, especially when the attacker is not provided with a complex computational infrastructure.

We evaluated our approach with respect to the security layer configurations described in Chapter 4. The features of the considered cryptosystems are summarized in Table 5.1. The WM_ECC_sk and WM_ECC_rc5 cryptosystems are both based on the WM-ECC library, used to execute key exchange and digital signature operations. They both perform symmetric encryption using respectively a Skipjack cipher with a 80 bit key and an RC5 cipher with a key of 160 bits. The TinyPairing cryptosystems is based on TinyPairing and uses a 208 bit key. In Table 5.1, the time needed to test a single key is reported for each cryptosystem, along with the maximum attack time, that is the time necessary to test all the possible keys. The reported elaboration time refers to the execution of the decryption operation on TelosB devices, characterized by a 4.15 MHz MSP430 microcontroller and a CC2420 radio chip and having a 10 kB internal RAM and a 48 kB program Flash memory.

It is important to point out that, due to the computational complexity of attacks, we will refer to simple attack scenarios where an attacker is able to gather partial information on the adopted cryptosystem but it does not have enough computational capabilities. As a consequence of these assumptions, in our experimental results, the resulting attack time will be significantly high but this will not affect the validity of the proposed approach as we are interested in illustrating how the probability of successfully completing an attack decreases.

We carried out our simulation experiments considering both worst and intermediate cases, and analyzed the cumulative distribution function (cdf) of the *attack time*. In both cases, we simulated an attacker sequentially exploring the key space.

Cryptosystem	*key len (bits)*	*time(ms)*	*max attack time(ms)*
WM_ECC_sk	80	0.001251	1,5123E+21
WM_ECC_rc5	160	0.001221	1,7845E+45
TinyPairing	208	13.019531	5,3560E+63

TABLE 5.1: Characteristics of the cryptosystems used in the experiments

We considered an observation interval as long as the attack time of the most complex cryptosystem, TinyPairing, and validity intervals of decreasing length. During an observation interval, we randomly generated 1000 different sequences of valid cryptosystems and recorded the time of successful attacks. The sequence length depends on the chosen validity interval.

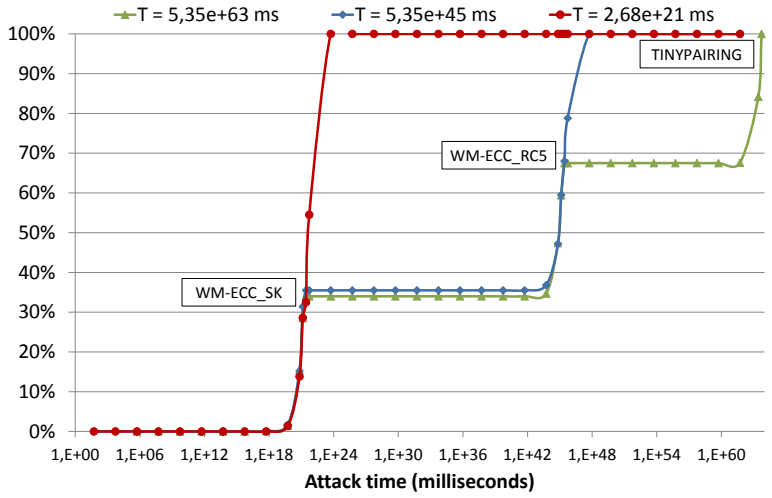

FIGURE 5.4: Worst case attack time cdf for large validity intervals

Figure 5.4 shows the attack time's cdf in the worst case; we chose three validity interval lengths in such a way to be comparable to the maximum attack times of the three different cryptosystems. The labels in the figure – note that the x-axis is on a logarithmic scale – identify three inflection points in the middle of the maximum attack times of each cryptosystem. These correspond to the maximum values of the attack time probability distribution functions (pdf) for each cryptosystem.

When analyzing the chart, a seemingly counterintuitive behavior can be identified: when considering smaller validity intervals the attacker seems to benefit. This is due to the fact that – in the considered scenario – when randomly choosing 1000 different cryptosystem sequences, the weakest cryptosystem WM_ECC_sk will be selected with 33% probability; in this case, considering the attack times reported

in Table 5.1, the attacker will always succeed for validity intervals longer than 1,5123E+21 milliseconds (worst case for WM_ECC_sk).

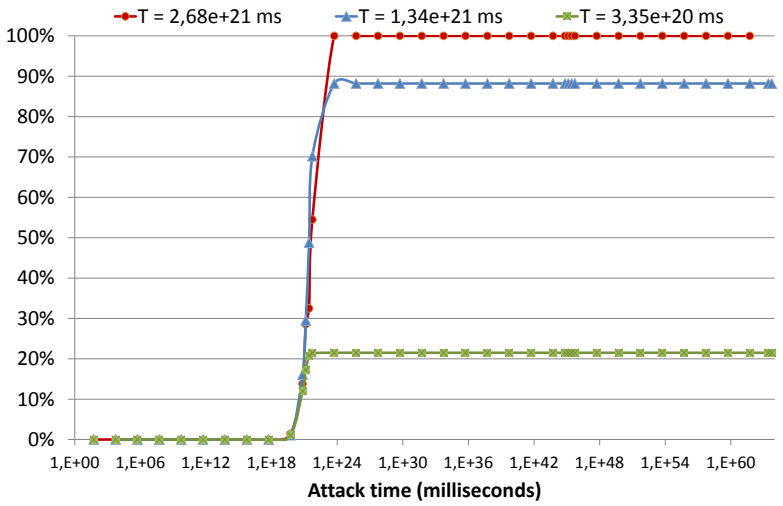

FIGURE 5.5: Worst case attack time cdf for short validity intervals

As illustrated in Figure 5.5, when further reducing the validity interval below the maximum attack time of the weakest cryptosystem, the trend changes and the attack time becomes higher, with the percentage of successful attacks reducing dramatically. The same behaviour is highlighted in Figure 5.6, that shows how the probability of completing a successful attack in a time t – with t equal to 5.36e+19, 1.34e+21, 2.68e+21 and 5.35e+45 milliseconds respectively – varies as the length of the validity interval changes: as soon as the validity interval goes below the maximum attack time of the weakest cryptosystem, the rate at which probability decreases becomes higher.

It is sufficient to consider validity intervals reasonably smaller than this critical value (threshold) to ensure both a low attack probability and a limited reconfiguration overhead. Due to the proposed attack model, the identified thresholds are significantly high but, as said, this does not affect the validity of the approach.

Analogous results can be obtained when reconfiguration is performed by selecting an equivalent cryptosystem that uses different parameters (i.e different keys). Figure 5.7 (a) shows the attack time cdf in the worst case when reconfiguration is performed by switching among three cryptosystems that implement the WM-ECC library with the Skipjack cipher, but have different keys. As shown, when

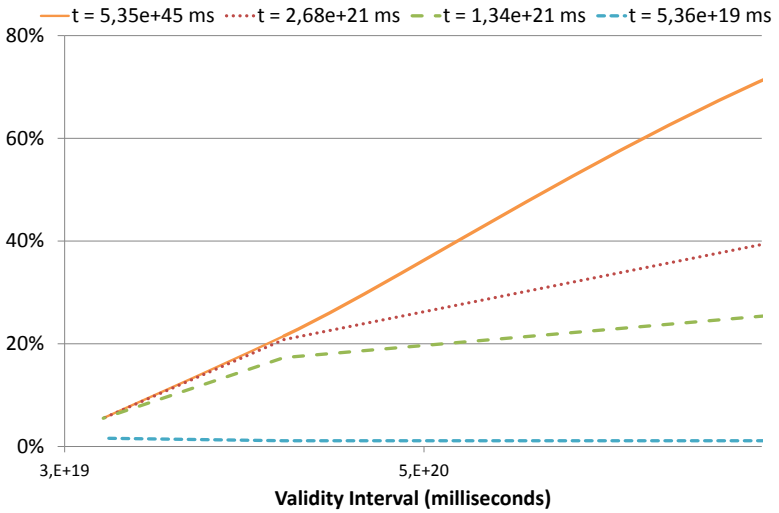

FIGURE 5.6: Probability of successful attack vs. length of validity interval

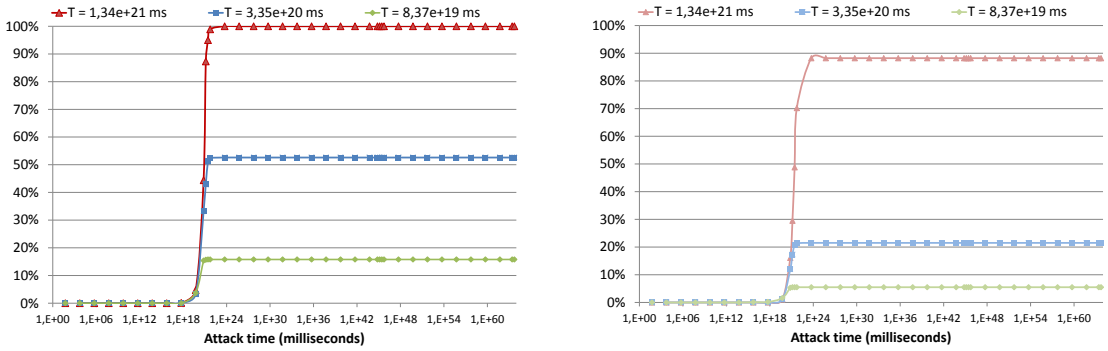

FIGURE 5.7: Worst case attack time cdf when (a) using the same cryptosystem with different keys - (b) using three different cryptosystems

reducing the validity interval, the probability of successfully complete an attack sensitively reduces as the intrinsic security level is restored each time a new key is activated. For comparison purposes, in Figure 5.7 (b) the attack time cdf of Figure 5.5 is reported; as expected, having more complex cryptosystems helps to achieve a lower probability of attack.

The worst case is very unlikely to occur. A more realistic assumption is that the attacker knows or is able to infer which kind of cryptosystem is currently used, but it does not know the key length associated with it. We assume that the attacker systematically tries all the possible keys for a given set of key lengths. Figure 5.8 compares the attack time cdf for the intermediate and the worst cases, under the assumption that the attacker performs a brute force attack using the set of key lengths in Table 5.2. As illustrated with respect to a validity interval of 5,36E+45

Cryptosystem	*key len (bits)*	*time(ms)*
WM_ECC_sk	[80]	[0.001251]
WM_ECC_rc5	[120,160]	[0.001120,0.001221]
TinyPairing	[180,208]	[11.023211,13.019531]

TABLE 5.2: Key lengths set

milliseconds, that is long enough to allow the attacker to break both WM_ECC_sk and WM_ECC_rc5 , the attacker's success probability is smaller in the intermediate case than in the worst case. Clearly, when the attacker's uncertainty about the used cryptosystem is higher, more key lengths will be tested, making the proposed approach even more effective.

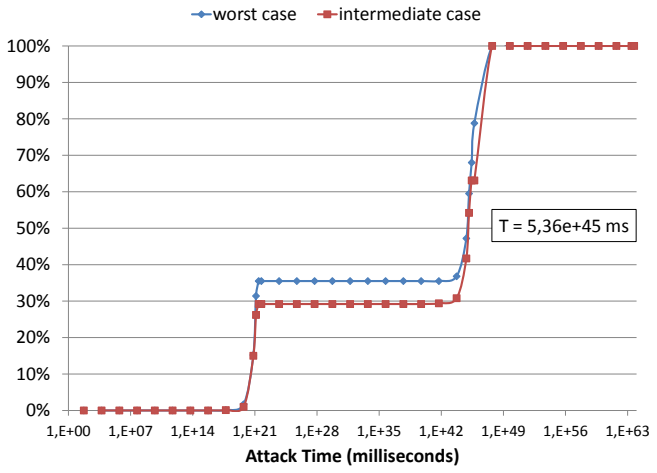

FIGURE 5.8: Comparison between worst and intermediate case for $T = 5,36E+45ms$

Chapter 6

Conclusion and future directions

In this thesis we addressed monitoring architectures composed of resource-constrained devices and developed a Moving Target Defense-inspired framework, based on reconfiguration at different granularity levels, to ensure security, performance and consumption requirements.

We formalized our reconfiguration approach by providing a reconfiguration model, identifying the reconfigurable parameters for a generic embedded node, a reconfiguration strategy for the selection of the new configuration to activate based on input requirements, and two different reconfiguration mechanisms referred to a WSN case study.

In order to define the reconfiguration strategy, we introduced a security metric based on attack coverage, able to measure the level of security provided by each configuration. Such metric, along with the commonly adopted performance and power consumption metrics, is used by the reconfiguration strategy to select the configuration that best meets the current quality requirements.

We developed a WSN case study to show the feasibility of the reconfiguration approach on real architectures and conducted theoretical and experimental analyses to prove its effectiveness in decreasing the probability of attack.

Although many interesting activities have been conducted with respect to the above discussed topics, several issues are still open and need to be investigated.

Our future plans include the design of a fully-automated reconfiguration strategy capable of identifying the system configuration that can best meet specific, dynamically changing requirements in terms of security, performance and power consumption. In order to do this, an innovative security metric for the comparison of different configurations must be defined; the most challenging aspect is the identification and modeling of the dependency relations existing between security and time, that constitutes a relevant and still unexplored topic.

We also plan to perform a deep evaluation of the optimal reconfiguration frequency and to introduce automatic mechanisms to map the existing requirements onto the available configurations (technological mapping).

Bibliography

[1] Pratyusa K. Manadhata and Jeannette M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37:371–386, 2011. ISSN 0098-5589.

[2] Executive Office of the President, National Science and Technology Council. Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program, December 2011.

[3] Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and Xiaoyang Sean Wang, editors. *Moving Target Defense - Creating Asymmetric Uncertainty for Cyber Threats*, volume 54 of *Advances in Information Security*. Springer, 2011. ISBN 978-1-4614-0976-2.

[4] Sushil Jajodia, Anup K. Ghosh, V. S. Subrahmanian, Vipin Swarup, Cliff Wang, and Xiaoyang Sean Wang, editors. *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, volume 100 of *Advances in Information Security*. Springer, 2013. ISBN 978-1461454151.

[5] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. In *Standards for Efficient Cryptography*, 2000. URL http://www.secg.org/download/aid-386/sec2-final.pdf.

[6] C.C. Tan H.Wang, B. Sheng and Qun Li. Wm-ecc: an elliptic curve cryptography suite on sensor motes. Technical Report WMCS-2007-11, October 2007.

[7] Xiaokang Xiong, D.S. Wong, and Xiaotie Deng. Tinypairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks.

In *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pages 1 –6, april 2010.

[8] David Evans, Anh Nguyen-Tuong, and John C. Knight. Effectiveness of moving target defenses. In *Moving Target Defense - Creating Asymmetric Uncertainty for Cyber Threats*, pages 29–48. 2011.

[9] Stephanie Forrest, Anil Somayaji, and David H. Ackley. Building diverse computer systems. In *Workshop on Hot Topics in Operating Systems*, pages 67–72, 1997.

[10] Pax team. URL http://pax.grsecurity.net/.

[11] Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 272–280, New York, NY, USA, 2003. ACM. ISBN 1-58113-738-9. doi: 10.1145/948109.948146. URL http://doi.acm.org/10.1145/948109.948146.

[12] Manuel Costa Jean-Philippe Martin Cristian Cadar, Periklis Akritidis and Miguel Castro. Data randomization. Technical report, Microsoft Research, 2008.

[13] Todd Jackson, Babak Salamat, Andrei Homescu, Karthikeyan Manivannan, Gregor Wagner, Andreas Gal, Stefan Brunthaler, Christian Wimmer, and Michael Franz. Compiler-generated software diversity. In Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, and X. Sean Wang, editors, *Moving Target Defense*, volume 54 of *Advances in Information Security*, pages 77–98. Springer New York, 2011. ISBN 978-1-4614-0976-2. doi: 10.1007/978-1-4614-0977-9_4. URL http://dx.doi.org/10.1007/978-1-4614-0977-9_4.

[14] D. Kewley, R. Fink, J. Lowry, and M. Dean. Dynamic approaches to thwart adversary intelligence gathering. In *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*, volume 1, pages 176 –185 vol.1, 2001. doi: 10.1109/DISCEX.2001.932214.

[15] M. Atighetchi, P. Pal, F. Webber, and C. Jones. Adaptive use of network-centric mechanisms in cyber-defense. In *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*, pages 183 – 192, may 2003. doi: 10.1109/ISORC.2003.1199253.

[16] S. Antonatos, P. Akritidis, E.P. Markatos, and K.G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Computer Networks*, 51(12):3471 – 3490, 2007. ISSN 1389-1286. doi: 10.1016/j.comnet.2007.02.006. URL http://www.sciencedirect.com/science/article/pii/S1389128607000710.

[17] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 127–132, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0. doi: 10.1145/2342441.2342467. URL http://doi.acm.org/10.1145/2342441.2342467.

[18] V. Casola, A. Mazzeo, N. Mazzocca, and V. Vittorini. A policy-based methodology for security evaluation: A security metric for public key infrastructures. *Journal of Computer Security*, 15(2):197–229, 2007.

[19] Common criteria project: Common criteria for information technology security evaluation 2.1. Technical report, US NIST, 1999.

[20] Trusted computer system evaluation criteria. Technical Report DoD 5200.28-STD, US Department Of Defense, 1985.

[21] Xiaohu Li, Timothy Paul Parker, and Shouhuai Xu. A stochastic model for quantitative security analyses of networked systems. *IEEE Trans. Dependable Sec. Comput.*, 8(1):28–43, 2011.

[22] Adam Barth, Benjamin I. P. Rubinstein, Mukund Sundararajan, John C. Mitchell, Dawn Song, and Peter L. Bartlett. A learning-based approach to reactive security. *IEEE Trans. Dependable Sec. Comput.*, 9(4):482–493, 2012.

[23] Peter Mell, Karen Scarfone, Sasha Romanosky, Peter Mell, Karen Scarfone, Sasha Romanosky, Carlos M. Gutierrez, and William Jeffrey Director. The common vulnerability scoring system (cvss) and its applicability to federal agency systems, 2007.

[24] Mohammad Salim Ahmed, Ehab Al-Shaer, and Latifur Khan. A novel quantitative approach for measuring network security. In *INFOCOM*, pages 1957–1965, 2008.

[25] Joseph Pamula, Sushil Jajodia, Paul Ammann, and Vipin Swarup. A weakest-adversary security metric for network configuration security analysis. In *QoP*, pages 31–38, 2006.

[26] Simon N. Foley, William Fitzgerald, Stefano Bistarelli, Barry OSullivan, and Mchel Foghl. *Principles of Secure Network Configuration: Towards a Formal Basis for Self-configuration*, volume 4268 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-47701-3.

[27] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[28] Alexander Becher, Er Becher, Zinaida Benenson, and Maximillian Dornseif. Tampering with motes: Real-world physical attacks on wireless sensor networks. In *Proceeding of the 3rd International Conference on Security in Pervasive Computing (SPC*, pages 104–118, 2006.

[29] G. Padmavathi and D. Shanmugapriya. A survey of attacks, security mechanisms and challenges in wireless sensor networks. *CoRR*, abs/0909.0576, 2009.

[30] T. Kavitha and D. Sridharan. Security vulnerabilities in wireless sensor networks: A survey. *Journal of Information Assurance and Security*, 5(1):31–44, 2010.

[31] Chris Karlof, Naveen Sastry, and David 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, New York, NY, USA, 2004. ACM. ISBN 1-58113-879-2. doi: 10.1145/1031495.1031515. URL http://doi.acm.org/10.1145/1031495.1031515.

[32] Mark Luk, Ghita Mezzour, Adrian Perrig, and Virgil Gligor. MiniSec: a secure sensor network communication architecture. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488. ACM Press, 2007. ISBN 978159593638X. URL http://dx.doi.org/10.1145/1236360.1236421.

[33] Shahin Farahani. *ZigBee Wireless Networks and Transceivers*. Newnes, Newton, MA, USA, 2008. ISBN 0750683937, 9780750683937.

[34] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8(5): 521–534, September 2002. ISSN 1022-0038. doi: 10.1023/A:1016598314198. URL http://dx.doi.org/10.1023/A:1016598314198.

[35] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21: 120–126, 1978.

[36] Ernest F. Brickell, Dorothy E. Denning, Stephen T. Kent, David P. Maher, and Walter Tuchman. Building in big brother. chapter SKIPJACK review: interim report, pages 119–130. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94441-9. URL http://dl.acm.org/citation.cfm?id=212412.212424.

[37] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. extended abstract in Crypto'01.

[38] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 514–532, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42987-5. URL http://dl.acm.org/citation.cfm?id=647097.717005.

[39] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *J. Cryptology*, pages 149–177, 2008.

[40] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press. ISBN 0-7803-9202-7. URL http://dl.acm.org/citation.cfm?id=1147685.1147768.

[41] Qiang Wang, Yaoyao Zhu, and Liang Cheng. Reprogramming wireless sensor networks: challenges and approaches. *Network, IEEE*, 20(3):48 – 55, May-June 2006.

[42] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, 2004. ISBN 1-58113-879-2.

[43] Jaein Jeong and D. Culler. Incremental network programming for wireless sensors. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 25 – 33, oct. 2004.

[44] Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P. Midkiff. Zephyr: efficient incremental reprogramming of sensor nodes using function call indirections and difference computation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 32–32, 2009.

[45] Wei Dong, Yunhao Liu, Chun Chen, Jiajun Bu, and Chao Huang. R2: Incremental reprogramming using relocatable code in networked embedded systems. In *INFOCOM, 2011 Proceedings IEEE*, pages 376–380, 2011. doi: 10.1109/INFCOM.2011.5935186.

[46] P.J. Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel. Tinycubus: a flexible and adaptive framework sensor networks. In *Wireless Sensor Networks, 2005. Proceedings of the Second European Workshop on*, pages 278 – 289, jan.-2 feb. 2005.

[47] W. Munawar, M. H. Alizai, O. Landsiedel, and K. Wehrle. Dynamic tinyos: Modular and transparent incremental code-updates for sensor networks. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –6, may 2010.

[48] V. Casola, A. De Benedictis, A. Drago, and N. Mazzocca. Analysis and comparison of security protocols in wireless sensor networks. In *Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on*, pages 52 –56, oct. 2011.

[49] P.K. Dutta, J.W. Hui, D.C. Chu, and D.E. Culler. Securing the deluge network programming system. In *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, pages 326 –333, 0-0 2006.