UNIVERSITÀ DEGLI STUDI DELL'AQUILA
DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATEMATICA (DISIM)

Dottorato di Ricerca in **Ingegneria e Scienze Dell'Informazione**

Curriculum **Systems Engineering, telecommunications and HW/SW platforms**

XXXII° ciclo

Titolo della tesi

A Security Framework for Wireless Sensor Networks

SSD **ING-INF/05**

Dottorando

Walter Tiberti

Coordinatore del corso

Prof. **Vittorio Cortellessa**

_____

Tutor

Prof. **Luigi Pomante**

_____

A.A. 2018/2019

A Framework for Wireless Sensor Network security

Extended abstract

Walter Tiberti
walter.tiberti@graduate.univaq.it
DEWS, University of L'Aquila, Italy

November 13, 2019

1 Introduction

Wireless Sensor Networks (*WSN*) are networks composed of small, wireless, battery-powered and sensor-equipped nodes, called *motes*. Thank to their characteristics, WSNs are often adopted as monitoring platforms in environments where the deployment of conventional networks are unfeasible. Typical applications of WSNs can be found in the *smart home* scenarios, in industrial applications, in safety-related and in military contexts.

The reference protocol stack for WSNs is the IEEE 802.15.4 [1], which defines the first two layers of the ISO/OSI network stack (PHY and MAC).

Providing security in WSNs is a non-trivial challenge: the performance limitations, the limited amount of storage and the finite energy resources cause the implementation of well-known state-of-art security solutions to be unfeasible. The IEEE 802.15.4 standard defines some security-related specifications, but they do not cover all the WSNs security issues.

In order to provide a complete solution for WSN security, we propose a *framework* to provide a set of lightweight security solutions which take WSN constrains into account. In particular:

- we address the *confidentiality* security requirement by mean of a set of lightweight cryptographic schemes based on *hybrid cryptography*, *Elliptic Curve Cryptography* (ECC) and new key transport protocols; at the same time, we introduce a hardware accelerator to overcome the performance limitations;
- we address the need for dynamic security checks on the WSN with the improvement of a misuse-based WSN *Intrusion Detection System*;
- we address the *integrity* requirements of both data and WSN motes by introducing a novel lightweight anti-tampering mechanism based on the *blockchain* technology;
- finally, we use an enhanced version of a famous mobile-agent middleware for WSN. The new version support additional features (e.g. energy-awareness) and integrate all the above techniques to build up a complete secure platform based on mobile agents for WSN.

2 Framework description

2.1 TAKS, ECTAKS and ECC-HAxES

TAKS [5][3][4] is a hybrid-cryptography scheme which uses *vector algebra* over a prime-field or a finite field to provide WSNs a lightweight yet effective solution for encryption and authentication. In our work, we extended TAKS implementation to support *star* and *cluster-tree* topologies. Also, we developed a version of TAKS suitable to be included as *Key Management Protocol* (KMP) in the recent IEEE 802.15.9 [2].

In order to increase the security level offered by TAKS, *ECTAKS* [6] has been introduced. ECTAKS combines the security of standard ECC protocols with the lightweight mechanism of TAKS. In our work, we provided a first real-word implementation of ECTAKS based on the TinyECC [12] library.

Finally, in order to overcome the performance limitation of public-key cryptography protocol implementations on WSN motes, in our work we designed and implemented a novel *hardware accelerator* for ECC-based protocols: the *ECC Hardware Accelerator for Embedded Systems* (*ECC-HAxES*). ECC-HAxES is an RTL design with minimum-area policy which has been designed to be configured on top of FPGA platforms and interfaced with a (master) embedded system (e.g. a WSN mote) to provide it with on-demand encryption (ECIES) and signature computation and verification (ECDSA).

2.2 WIDS and TinyWIDS

WIDS [7][3][4] is a misuse-based intrusion detection system for WSN which uses *Weak Process Models* (a variant of the Hidden Markov Models) to estimate and track the *state* of a WSN mote from a series of of *observable* events. In our work, we enriched WIDS and provided a new TinyOS-based implementation called *TinyWIDS* [8].

2.3 Agilla evolution

Agilla [10] is a TinyOS-based mobile-agent middleware (MAMW) for WSN. In Agilla, software applications are deployed as *agents* which can move or clone themselves across the WSN. We ported Agilla [11] to the newer TinyOS platforms.

2.4 Anti-tampering solution

In order to address both the data security and the *node capture* attacks, we developed *WSN-LBC*, a novel lightweight blockchain-based anti-tampering mechanism for WSN monitoring applications. WSN-LBC uses a ledger composed of a selectable number of blockchains with different reliability levels. Upon reception of data messages, the receiver (usually the coordinator) can check the validity of the message and decide (depending of the message itself and the reliability associated to the sender node) whether the data should be stored in one of the blockchains or refused. The coordinator also update the reliability associated to the sender so that a good-behaving node will always see its data stored in high-reliability blockchains, while bad-behaving nodes will be, after a selectable number of invalid messages, forbidden entirely from the WSN communication. This help solve *node capture* attacks since, when a captured node is removed from the WSN, it will lose the ability to produce valid messages (e.g. failing to produce the correct hashes required). Once the captured node is re-injected into the WSN, this will make it to lose its reliability and, eventually, causing the coordinator to systematically discard its messages.

2.5 Security Framework

The proposed security framework integrate all the previously described components into a new lightweight environment for developing secure WSN applications. Starting from [9], we adopted the new version of Agilla and integrate it with the new TAKS-based scheme (i.e. ECTAKS) used as KMP in the IEEE 802.15.9-based stack. We added the support for using ECC-HAxES and added TinyWIDS as background intrusion detection system. Finally, the WSN-LBC mechanisms has been added as optional anti-tampering mechanism.

References

[1] IEEE Standard for Low-Rate Wireless Networks," in IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011) , vol., no., pp.1-709, 22 April 2016 doi: 10.1109/IEEESTD.2016.7460875

[2] IEEE Recommended Practice for Transport of Key Management Protocol (KMP) Datagrams," in IEEE Std 802.15.9-2016 , vol., no., pp.1-74, 17 Aug. 2016 doi: 10.1109/IEEESTD.2016.7544442

[3] Pugliese M., "Managing Security Issues in Advanced Applications of Wireless Sensor Networks", PhD Thesis, 2008

[4] Marchesani S., "A Middleware approach for WSN security: Cryptography and Intrusion Detection for Real-World Application", PhD Thesis, 2013

[5] M. Pugliese and F. Santucci, "Pair-wise network topology authenticated hybrid cryptographic keys for Wireless Sensor Networks using vector algebra," 2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, Atlanta, GA, 2008, pp. 853-859. doi: 10.1109/MAHSS.2008.4660137

[6] Pugliese, Marco & Pomante, Luigi & Santucci, Fortunato. (2012). Secure Platform Over Wireless Sensor Networks. 10.5772/34607.

[7] Pugliese, Marco & Giani, Annarita & Santucci, Fortunato. (2009). Weak Process Models for Attack Detection in a Clustered Sensor Network Using Mobile Agents. Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. 24. 33-50. 10.1007/978-3-642-11528-8_4.

[8] Bozzi Luciano , Giuseppe Lorenzo , Pomante Luigi , Pugliese Marco , Santic Marco , Santucci Fortunato & Tiberti Walter. (2018). TinyWIDS: a WPM-based Intrusion Detection System for TinyOS2.x/802.15.4 Wireless Sensor Networks. 13-16. 10.1145/3178291.3178293.

[9] L. Pomante, M. Pugliese, S. Marchesani and F. Santucci, "WINSOME: A middleware platform for the provision of secure monitoring services over Wireless Sensor Networks," 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), Sardinia, 2013, pp. 706-711. doi: 10.1109/IWCMC.2013.6583643

[10] C. L. Fok, G. C. Roman, and C. Lu, "Agilla: A Mobile Agent Middleware for Self-Adaptive Wireless Sensor Networks", ACM Transactions on Autonomous and Adaptive Systems, Vol. 4, No. 3, Article 16, 2009

[11] L. Corradetti, D. Gregori, S. Marchesani, L. Pomante, M. Santic and W. Tiberti, "A renovated mobile agents middleware for WSN porting of Agilla to the TinyOS 2.x platform," 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), Bologna, 2016, pp. 1-5. doi: 10.1109/RTSI.2016.7740615

[12] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," 2008 International Conference on Information Processing in Sensor Networks (ipsn 2008), St. Louis, MO, 2008, pp. 245-256.

UNIVERSITÀ DEGLI STUDI DELL'AQUILA
DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATEMATICA (DISIM)

Dottorato di Ricerca in **Ingegneria e Scienze Dell'Informazione**

Curriculum **Systems Engineering, telecommunications and HW/SW platforms**

XXXII° ciclo

Titolo della tesi

A Security Framework for Wireless Sensor Networks

SSD **ING-INF/05**

Dottorando

Walter Tiberti

Coordinatore del corso

Prof. **Vittorio Cortellessa**

_____

Tutor

Prof. **Luigi Pomante**

_____

A.A. 2018/2019

Contents

List of Figures

Chapter 1

Introduction

1.1 Context

Today, after more than 40 years of following the Moore Law, the modern computing platforms have incredible computational power, almost endless memory storage and they are reaching the point at which the physics (in the form of heat dissipation, energy limitations, etc.) start to pose non-trivial issues to technology improvement.

With the introduction of the *Internet of Things* paradigm, in some fields of technology the form factors, the amount of consumed energy and the flexible and innovative applications, made the performance and memory storage aspects less important. Among such platforms, the *Wireless Sensor Networks* (WSN) represent a core technology for distributed monitoring applications. WSN uses nodes (often called *motes*) which are designed and constructed to provide a very low-energy platform capable of retrieving sensor measurements and forwarding them to more powerful platforms. The absence of a wired link across the nodes makes the WSNs a very flexible platforms which can be adopted in almost any context, e.g., industrial monitoring, smart grids [123], battlefield monitoring [124] etc. Comprehensive lists of possible WSN applications can be found in [125][126].

Flexibility comes at a cost: WSN nodes are usually powered by batteries and their performance, memory and capabilities are very limited in favor of a prolonged battery life. As a consequence, the software running on WSN needs to be as much optimized as possible. Moreover, in order to further reduce the impact on performance and on memory, all those features which are not functional requirements are usually removed. *Security*-related features are one example of those features.

Providing a good level of security in resource-constrained platforms like WSNs is a non trivial issue. From one side, those platforms are more prone to attacks due the lack of a full networking stack including dedicated security layers; from the other side, the lack of computational resources causes developers to

decide whether include security features or leave more space for the applications. For example, providing *confidentiality* in transmissions though a cryptographic scheme implies a considerable overhead in latencies and in throughput, additional space required for encryption/decryption code and data and additional requirements such as *key management and distribution*, *secure random number generators*, etc.

Luckily, the research community proposed and still propose new ad-hoc techniques to provide security features in resource-constrained platforms. Those techniques are developed so that it is possible for the target resource-constrained platforms to have a valuable security level while being lightweight on the computation and the memory storage required.

However, due to the heterogeneity of the platform involved, little efforts have been made to provide a full security solution which can both grant security features to resource-constrained platforms and be flexible enough to be adapted to the requirements and limitations of the target application.

1.2 Objectives

The objective of this thesis is to define a *security framework* composed of tools, (*agent-based WSN middlewares*, *intrusion detection systems*) techniques (*hybrid cryptography*, *elliptic curve cryptography*, *hardware acceleration for cryptography*, *blockchains as anti-tampering mechanism*) and methodologies (*model-based porting for WSN software applications*) specifically designed for the resource-constrained devices, with particular focus on *Wireless Sensor Networks*. In order to do so, the thesis objective has been split into the following *three* sub-objectives:

- **Objective 1**: to analyze existing state-of-the-art techniques and tools and *test*, *adapt* and *enhance* them to create new versions able to provide additional and/or empowered features.

- **Objective 2**: to study *new techniques* which could be useful in the context of resource-constrained platforms; then to design *new tools and components* which exploits such new techniques while fulfilling the requirements and limitation of the target platforms.

- **Objective 3**: to provide all the necessary abstractions and components to *adapt* and *harmonize* the components to provide a set of tools which can be flexibly combined together to offer the security features required. Considering the work described in [12], the final objective is to use the defined security framework and its components to realize an updated, full-featured *security platform* for Wireless Sensor Networks which could be exploited in *real* scenarios and *real* Wireless Sensor Network node platforms.

1.3 Thesis Contributions

The main contributions provided by the research activities described in this thesis are the following:

- Refinement and re-development of the *TAKS*[19] cryptographic scheme
- The implementation of the refined TAKS on the TinyOS operating system.
- The definition and development of a IEEE 802.15.9 [17] compliant version of TAKS.
- The definition and development of a Elliptic Curve Cryptography version of TAKS: *ECTAKS*.
- The definition and development of a elliptic curve cryptography hardware accelerator on reconfigurable platforms for embedded systems: *ECC-HAxES*.
- The refinement and development of WIDS intrusion detection system for WSN: *TinyWIDS* [1].
- The porting and enhancement of the Agilla Middleware [34]: *Agilla2*.
- The definition and development of a blockchain-based anti-tampering mechanism for WSN: *WSN-LBC*.
- The integration of TAKS into the *DSME* IEEE 802.15.4 [16] MAC Behavior (*DESTAK*) and the development of an Omnet++ based simulator for validation and performance evaluation
- Contribution to the following research projects (and related deliverables):
  - *European*: VISION, SafeCOP
  - *National*: SEAMLESS

1.4 Thesis Organization

This thesis is organized in three parts. In Part I, the basic knowledge and software tools required for a quick understanding of the presented research activities are described. In particular, Chapter 2 analyzes the main Wireless Sensor Network architectures, the hardware and the communication protocols; Chapter 3 and 4 focus on the software environments in which WSN applications are developed. Existing security measures are analyzed in Chapter 5, with a focus on cryptography in Chapter 6 and Chapter 7 while Intrusion Detection Systems are described in Chapter 8.

Part II presents the research activities that represent the core of this Thesis. An overview on the proposed contributions is provided, describing also how they have been integrated to form a complete security framework (Chapter 9).

This Part contains the TAKS analysis in Chapter 10, ECTAKS in Chapter 11, ECC-HAxES in Chapter 12, WIDS and TinyWIDS in Chapter 13, the Agilla Middleware in Chapter 15 and WSN-LBC in Chapter 14.

Part III brings the attention on some important use cases (related to several research projects) which have seen the exploitation of the results of our research activities, representing also a *de-facto* validation. In Chapter 16, 17 and 18 National and Europeans project contributions are presented, while in Chapter 19 a joint work in the context of an industrial application is presented.

Finally, Part IV concludes this work with a resume on the provided contributions and an overview on the future works and improvements.

Part I

Background

Chapter 2

IEEE 802.15.4-based Wireless Sensor Networks

This chapter introduces the IEEE 802.15.4-based Wireless Sensor Networks. In Section 2.1 the main concepts and features of Wireless Sensor Network are described while in Section 2.2 a brief introduction on the IEEE 802.15.4 standard is reported. The chapter ends with a description of the additional layers which, together with the IEEE 802.15.4 standard, forms a complete networking stack.

2.1 Wireless Sensor Networks

A *Wireless Sensor Network* (abbr. **WSN**, Figure 2.1) is a network composed of nodes (often called *motes*) equipped with a set of sensors and interconnected by means of a wireless medium such as radio waves.

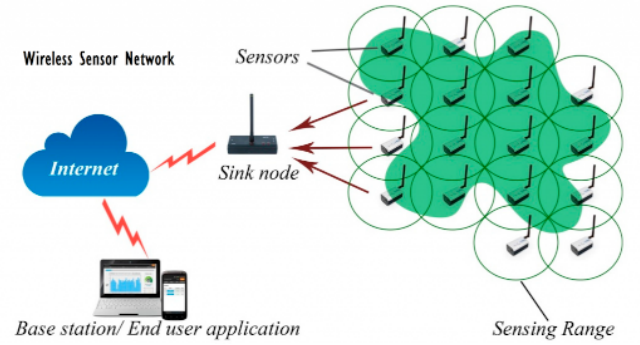


Figure 2.1: A WSN

WSNs can be adopted in various context and applications, although WSNs find their optimal usage in monitoring applications, where one or more environ-

mental phenomena needs to be measured across an area in which it is difficult or unpractical to deploy a conventional wired network. WSNs cover also an important role in *industrial* applications as a tool to monitor and control industrial processes.

Typical WSN applications are found also in the so-called *smart home* systems, e.g., to perform distributed monitoring, to control other devices or as a mean to provide small-range communication capabilities in the whole environment.

WSNs are used also in application domains where the *public safety* is concerned, from public roads safety (e.g., [98]) to wildfire monitoring (e.g., [103]) or as landslide prevention and monitoring platform [15].

Finally, WSNs find application in military contexts e.g., battlefield monitoring, supply storage, management and localization, soldiers coordination or augmented reality applications.

Figure 2.2 shows a summary of the common WSN applications.

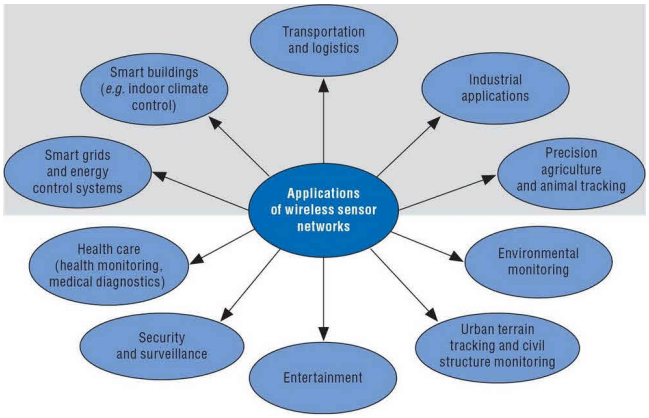


Figure 2.2: WSN applications

2.1.1 Main Features

Thanks to the combination of a computing platform, sensors (sometimes also actuators) and one or more RF transceivers, WSN motes and the WSNs themselves have a set of peculiar features that make them a solid choice for a wide range of contexts.

The main features of WSNs are briefly described below. The *deployment* and *installation* times of WSNs are greatly reduced, since the motes are usually light, small and require no or light maintenance once operational, e.g., minimal deployment scenario could consist in just the dissemination of the WSN motes in the area of interest, with no need for configuration, connection-checking, etc.

Thanks to these features, WSNs are *scalable* in size and they are the platform of choice in applications requiring dense (or redundant) monitoring capabilities,

since the WSN communication protocols are designed to allow WSN nodes to *associate/disassociate* from a WSN dynamically and it is trivial to increase the number of WSN nodes while keeping the WSN operational.

The overall *costs* are also reduced: the WSN motes are less expensive than other solutions and there is no need for any special equipment to start using them. In fact, a large number of software applications are already available and free to be used without charge. Also, in typical WSN, the radio frequencies adopted in the inter-communication are among the unlicensed spectrum bands (*ISM bands*), so no license has to be acquired and there are no costs involved.

The WSN motes are usually *battery-powered* to avoid being dependent on electric power grids. However, this feature highlights an important constraint of WSN nodes: the *energy consumption*. Since the energy available to WSN nodes comes from a fixed capacity battery, the hardware platforms and the software applications are designed to be energy-efficient, so that a careful usage of the available energy results in a prolonged WSN mote battery life.

2.1.2 Hardware

From the hardware point-of-view, WSNs can differ in many aspects. In fact, this is another characteristic of WSN: the *heterogeneity* of the hardware platforms. It is common for a WSN to have different classes of devices running and communicating together at the same time with the same communication stacks. While heterogeneity of the hardware platforms can be seen as an advantage in term of costs and upgradeability, this characteristic of WSN often causes software to be more complex.

Despite of the differences of the platforms themselves, the jargon[1] adopted in the context of WSNs is similar. Below, a list of common terms is presented:

- **Mote** - a (generic) WSN node;
- **Personal Area Network (PAN) Coordinator** - the WSN node responsible of the coordination of all or a part of the WSN; in some network topologies, there can be various *local* Coordinators and a "master" Coordinator.
- **Reduced Function Device (RFD)** - a mote capable of performing only basic tasks as sensor reading and communication.
- **Full Function Device (FFD)** - a WSN node with enough computational power to perform additional computation apart from the basic tasks of a RFD. The PAN Coordinator role needs to be covered by a FFD. Every RFD or FFD other than the Coordinator is also often called simply **device** node.
- **Cluster** - a part of a WSN consisting of a group of motes and a local coordinator. The coordinator of a cluster is also called **Cluster Head**.

[1] Some of the terms are taken directly from the IEEE 802.15.4, which is discussed in 2.2

Hardware: Micro-controllers

Each WSN mote has usually one *micro-controller* unit (**MCU**). A typical MCU for WSN is a Harvard-based low-cost 8-bit or 16-bit MCU with relevant features in energy management (e.g., the ability to power down single parts of the MCU in order to reduce energy consumption).

Examples of common MCUs are the *MSP430* [21] from Texas Instruments, the *ATMega* family of MCU [22] such as *ATMega128* or the *ATMega328p*.

In Table 2.1 a comparison of the most famous and adopted MCU for WSN motes is presented.

	TI MSP430 F1611	TI MSP430 F2618	Atmel ATMega 128	Atmel ATMega 1281
Bits	16	16	8	8
CPU/MCU Freq.	8 MHz	16 MHz	16 MHz	16 MHz
Flash Memory	48 KB	116 KB	128 KB	128 KB
RAM	10 KB	8 KB	4 KB	8 KB
Energy consumption	330 μA	365 μA	500 μA	500 μA

Table 2.1: Comparison of common WSN MCUs

The second hardware core component of a WSN mote is the *radio transceiver*, usually deployed in the form of a *radio chip* and an external antenna. The radio chips adopted are various, but one of the most used in WSN motes is the *CC2420*[23] from Texas Instruments. This radio chip is present in numerous platforms and many communication stacks are designed to use its features. Atmel-based WSN motes optionally adopt the *AT89 RF230*[24]. A brief comparison of these two transceiver is reported in Table 2.2.

	TI CC2420	Atmel AT89RF230
IEEE 802.15.4 compliant?	Yes, with issues [31]	Yes
HW Encryption?	Yes, AES 128bit	No
Energy Consumption	~17 mA	~16 mA
Communication with MCU	SPI	SPI

Table 2.2: Comparison of common WSN RF transceivers

Another component of WSN motes is worth additional considerations: the available *memory storage*. In fact, this can represent one of the main constraint when developing applications on top of a WSN mote. The storage available to a mote can take different forms, usually:

- Internal (*Flash*) memory, located inside the MCU, is used mainly for storing the code of the applications;
- *RAM* memory, used as work memory. Application data is stored in RAM;

- *EEPROM* memory, used as additional storage. It can be *internal* or *external* (i.e. off-chip). In newer platforms, the external EEPROM memory is replaced with an equivalent Flash memory.

An exhaustive list of WSN mote platforms is reported in [39].

Finally, the last core component of a WSN mote is the available set of *sensors*. Depending on the platform, the sensors can be found directly on the mote board or as a separate board (i.e. the *sensor-board*) which can be connected to the mote expansion ports. Common sensors found in WSN mote platforms are *light*, *temperature*, *humidity* sensors, *accelerometers*, *magnetometers*, *gyroscopes* and *GPS modules*.

In Table 2.3 a number of commonly used WSN hardware platforms are listed and compared.

Model	MCU	RF Chip	Storage (Flash/RAM/	Sensors
CM5000 (telosb)	MSP430 F1611	CC2420	48KB 10KB 1MB	Temp.+Hum/Light
XM1000	MSP430 F2618	CC2420	116KB 8KB 1MB	Temp.+Hum/Light
MicaZ	ATMega128l	CC2420	128KB 4KB 512KB	(via sensorboards)
Iris	ATMega1281	RF230	128KB 8KB 512KB	(via sensorboards)

Table 2.3: Comparison of common WSN hardware platforms

An image of a *telosb* mote is shown in Figure 2.3.

2.1.3 Software

The software part of a WSN mote is usually subdivided into layers, depending on the final application of the WSN mote.

First of all, the *protocol stack* is a set of layers which takes care of interfacing with the radio chip to achieve physical radio communication and providing user-level applications a series of common networking interfaces (e.g., ISO/OSI protocol stack) which can be used to send arbitrary application-specific data. Usually, a minimal but functional IP-based protocol stack implementation is provided. In particular, the software implementation starts with the radio chip interface (which provides the PHY layer and the lower part of the MAC layer), then the upper part of the MAC layer (e.g., de/association, channel scheduling, etc.). The upper layers (NWK, TSP and APL) are all software-based and, if present, they are implemented with common solutions, described in the further sections.

A second aspect of WSN mote software is the *low-level infrastructure*. Due the differences in the hardware platforms, this part of software is usually strongly

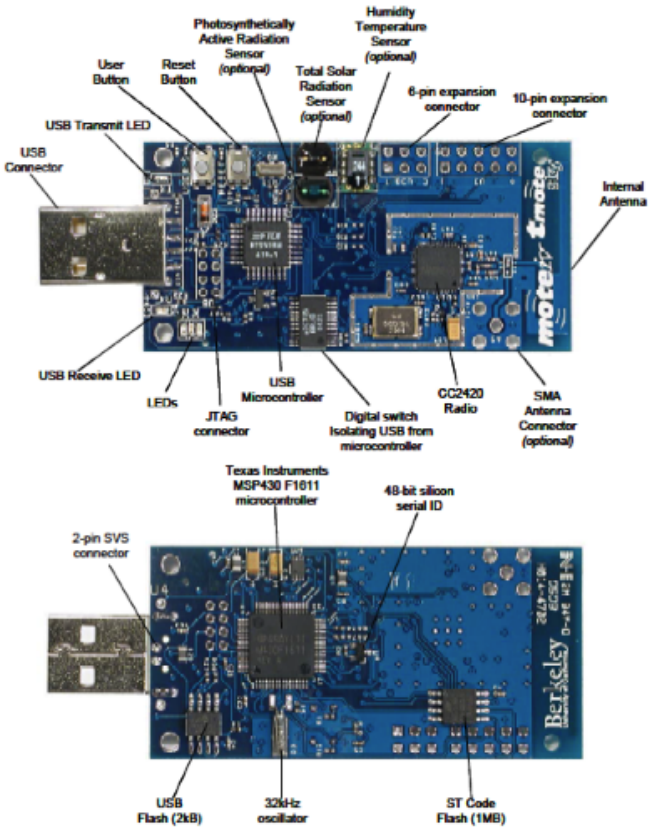


Figure 2.3: telosb-like WSN mote

hardware-dependent. As a consequence, software developers need to adopt hardware-abstraction libraries or provide different implementations of the same application for each given hardware platform.

The former is the preferred approach, since it allows developers to focus on the final application functionalities instead of the low-level hardware details. In order to support this approach, a number of different frameworks (sometime improperly called *operating systems*) have been proposed. *TinyOS* [30] is one of the most famous framework for WSN; it is examined with more details in Chapter 3. *Contiki* and its evolution, *Contiki-ng* [36] is an alternative OS for WSN platforms. It focuses on providing IPv6 and TSCH [25] support. Contiki-ng support different WSN mote platforms and some 32-bit ARM-based platforms. *RiotOS* [37] is a recent OS, targeting not only WSN platforms but also generic *Internet-of-Things* platforms. It focuses on real-time capabilities, lightweight inter-process communication and a solid hardware abstraction layer.

Despite using a framework allows to reduce the hardware-dependency of low-

level software layers, sometime such abstractions are not enough to provides a comfortable environment for software development. In such cases, an additional software layer can be adopted, a *Middleware* (MW). A MW is a software layer that stands in the *middle* between the low-level software layers (e.g. frameworks, OSes, etc.) and the final application layer. A MW allows developer to raise the abstraction level and provide powerful software primitives, usually oriented to the final application. A taxotomy of modern MWs for WSN is reported in [40]. Also, in Section 4.2, a particular class of MW (*Mobile-Agent Middlewares*) is discussed and analyzed.

2.2 IEEE 802.15.4 Standard

The **IEEE 802.15.4** [16] is the reference standard for the low-rate wireless personal area networks (LR-WPAN), the class of networks in which WSNs is widely considered to belong.

The IEEE 802.15.4 standard describes the first two layers of the ISO/OSI network stack (PHY and MAC). Each layer provides a set of services (Figure 2.4 via the so-called *service access points* (SAP). For the PHY and the MAC layer, the SAP exposed to the upper layers are two: one used to provide management primitives (the *Physical Layer Management Entity SAP, PLME-SAP* and the *MAC Layer MLME-SAP*)

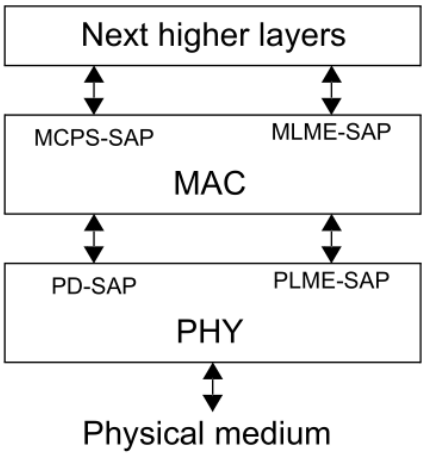


Figure 2.4: IEEE 802.15.4 layers service access points

The standard defines two types of network nodes: the *Full Function Devices* and the *Reduced Function Devices*. Only the FFDs could take the role of *PAN Coordinator*.

The two main topologies explicitly supported by the standard, apart from the basic point-to-point connection, are the *star* topology (i.e. a collection of point-to-point connections with a single coordinator) and the *Cluster-Tree* topology.

The latter consists in a set of *clusters*, each logically grouping a set of nodes under a *Local PAN Coordinator* which, in turn, is connected to a hierarchically higher level of connections, ending with a main, global WPAN Coordinator for the whole WPAN. Figure 2.5 shows the two topologies.

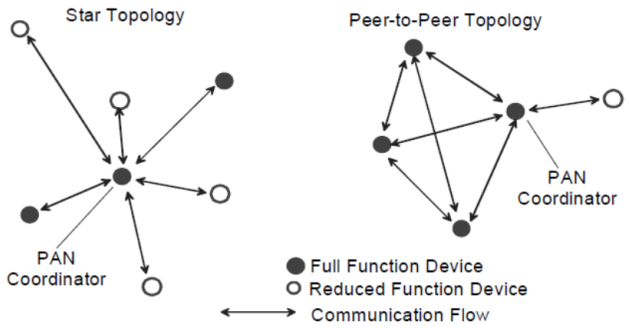


Figure 2.5: IEEE 802.15.4 supported network topologies

The IEEE 802.15.4 standard PHY layer uses the frequency bands shown in Figure 2.6.

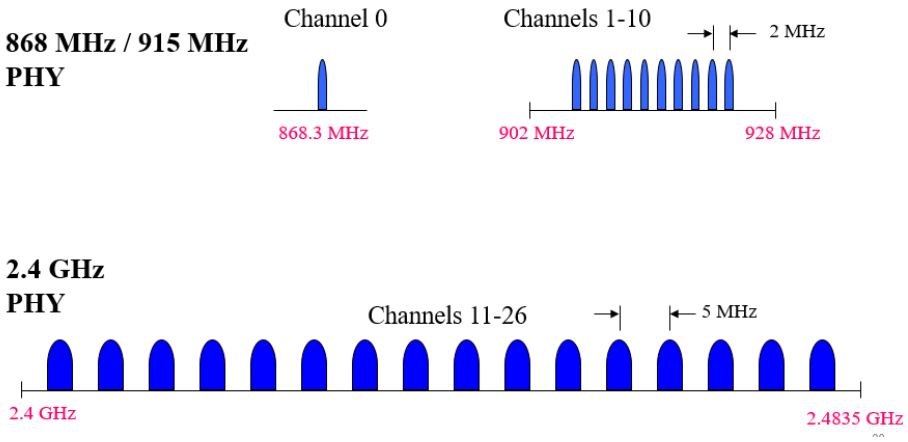


Figure 2.6: IEEE 802.15.4 spectrum bands

2.2.1 Channel Access

The IEEE 802.15.4 WPANs can choose between two mode of operations: the *beacon-less* and *beacon-enable* mode. In the former, communications (and channel accesses) happen asynchronously and the channel access mechanism adopted is the CSMA-CA. In the latter, a special periodic message (the *beacon*) emitted

by the Coordinator dictates when the communication can happen. In particular, the beacons define a *superframe* structure (Figure 2.7).

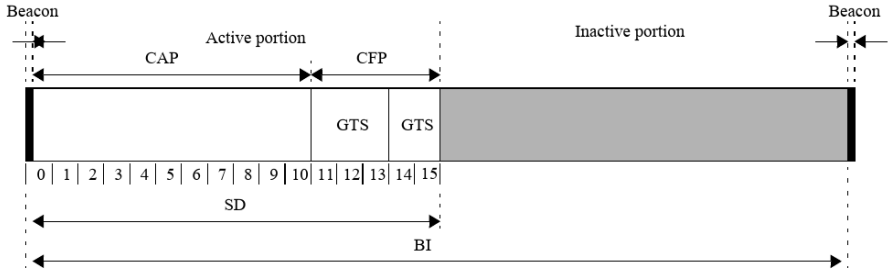


Figure 2.7: IEEE 802.15.4 Superframe structure

The IEEE 802.15.4 superframe is divided into an *active* portion, when the communication can happen, and an *inactive* portion, where nodes can *sleep* and save energy because no communication should happen.

The active portion is also divided in two *periods*. The first, the *Contention Access Period* (CAP) is where the channel access is *contended* by nodes and regulated by the CSMA-CA mechanism. The second period, called *Guaranteed Time Slots* (GTS) is a optional period in which a set of nodes (or even all) gets a specific per-node time slot which can be used to access the channel without competitors.

2.2.2 Frame Structure

The the complete frame structure for an IEEE 802.15.4 is shown in Figure 2.8.

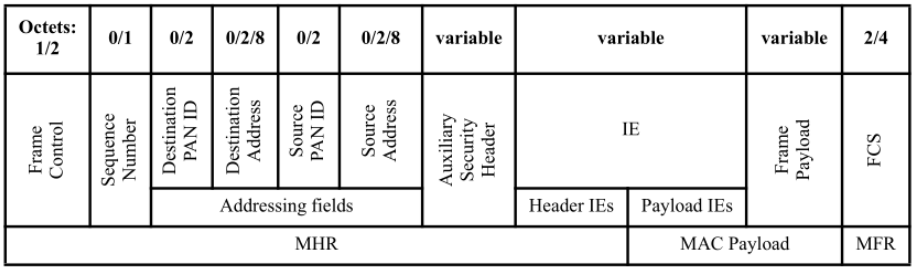


Figure 2.8: IEEE 802.15.4 frame format

Starting from a bird eye view of the structure, the IEEE 802.15.4 frame is divided into a **MAC Header** (MHR), a **MAC Payload** and a **MAC Footer** (MFR).

The MAC Header contains meta-information about the frame itself and the so-called *addressing fields*, i.e., the source/destination mote addresses of the

mote and of the WPAN. The addresses can be **short** (16 bit) or **long** (64 bit). The size can be chosen depending on the number of nodes in the WPAN.

The MAC Header contains also some security-dedicated fields and extensions, which are analyzed in details in Section 5.1.

The MAC Footer consist of a field with link quality information, a field for a 8-bit *Received Signal Strength Indicator* (RSSI) and the 16-bit ITU-T *Cyclic Redundant Code CRC* [26] (or, optionally, the ANSI X3.66-1979 32-bit version) used by receivers to detect and eventually correct bit errors in the frame. The CRC is computed on MHR and on the Payload using the polynomial (the *generator*)

$$x^{16} + x^{12} + x^5 + 1$$

.

Despite the absence of any restriction about the content of the MAC Payload, it deserves some additional considerations: since the full PSDU size for a frame is 127 byes, depending on which field is used or omitted in the MHR, the available size as MAC Payload is often limited to be half or less bytes long. A practical MAC Payload size, considering only the mandatory fields in the MHR and MFR, could be **100** bytes. Also, when using additional fields and the *Payload Information Elements* (Payload IE, see Chapter 19), the space available as MAC Payload can quickly be exhausted.

2.2.3 IEEE 802.15.4e Standard and MAC behaviors

The first version of the IEEE 802.15.4 standard has been released in 2008. The following years, a number of revisions have been released. As the time of writing, the last standard revision is the IEEE 802.15.4-2015.

One of the reason behind the release of multiple revisions of the standard is due to the merge of parallel standards targeting and providing adaptation of the IEEE 802.15.4 to specific environments. This is the case of the *IEEE 802.15.4e* standard.

The IEEE 802.15.4e standard was a proposed amendment aimed to provide some additional features in the direction of the industrial applications. The proposal has been merged into the recent versions of the IEEE 802.15.4 in the form of new *MAC behaviors*. Examples of well-known MAC behaviors for industrial application are the *Time-Slotted Channel Hopping* (TSCH) and the *Deterministic Synchronous Multichannel Extension* (DSME). The latter is reviewed in Chapter 19.

2.3 ZigBee

The IEEE 802.15.4 provides standardized PHY and MAC layers, with no additional information on upper layers. **ZigBee** is a protocol that uses IEEE 802.15.4 standard as lower layers (Figure 2.9) and provides a set of upper layers to be used in the IoT domain.

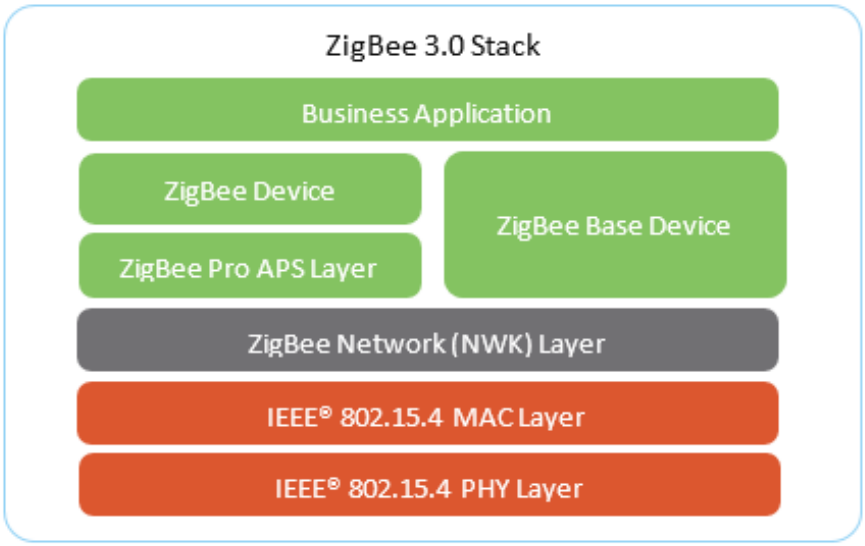


Figure 2.9: ZigBee

ZigBee stack specification is maintained by the *ZigBee Alliance* [73], a consortium of tens of industrial companies which take care of the ZigBee releases (the current version is the **3.0**) and the *certification* of products.

ZigBee defines three classes of nodes:

- the *ZigBee Coordinator* (ZC): the device which acts as Coordinator and gateway to external networks;
- the *ZigBee Routers* (ZR): the devices which route and forward packets;
- the *ZigBee End-devices* (ZED): the devices which perform basic tasks and communicate only with ZR or the ZC.

Figure 2.10 shows the evolution of the classic IEEE 802.15.4 topologies and the *Mesh* topology (defined in ZigBee).

2.4 IoT protocol stack

This section analyze some common techniques, methods and protocols which are adopted to enhance the communication capabilities and experience within IEEE 802.15.4 WSNs.

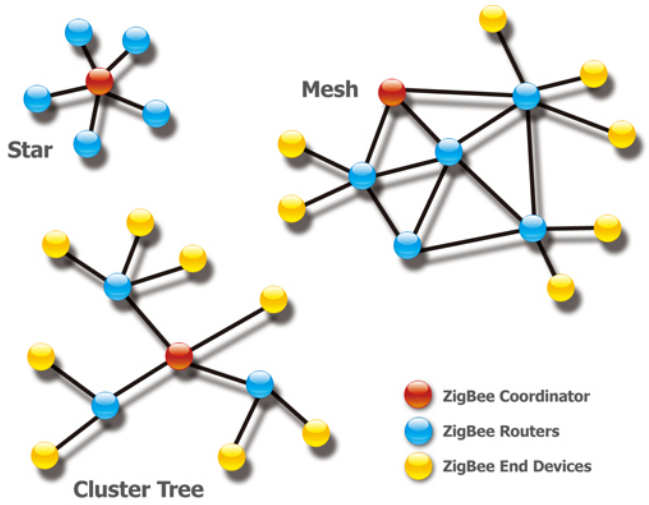


Figure 2.10: ZigBee topologies

2.4.1 6loWPAN

An attempt to bridge the gap between the conventional TCP/UDP-IP networks with IEEE 802.15.4 WPANs is the **6LoWPAN** specification. 6LoWPAN defines a set of techniques to embed classical IPv6 packets in IEEE 802.15.4 data frames. It is defined in the RFC 4944 and further extended in later RFCs [27].

One of the techniques 6LoWPAN adopts to adapt IPv6 packets is to use Link-local IPv6 addresses (e.g., `FF80::xxxx:xxxx:xxxx:xxxx`) as WPAN node IPv6 address, where the `x`s are the 64-bit EUI-64 MAC Extended Address of a mote. The RFC 4944 defines various other techniques to adapt or construct valid IPv6 addresses from IEEE 802.15.4 long/short addresses and to simulate the IPv6 Multicast capabilities on top of IEEE 802.15.4 WPANs.

Since the available MAC payload of IEEE 802.15.4 is limited to ∼80 bytes, 6LoWPAN proposes various *compression* techniques to increase the space available for applications. Moreover, it requires the MAC layer to support the *Fragmentation* and *Reassemble* of data transmissions.

From the *security* point-of-view, 6loWPAN offers no additional features. In fact, common attacks such as *address duplication* are still possible (see Section 13 in [27])

2.4.2 ROLL

In *Low-Power and Lossy Networks* (LLN) such as WSNs, implementing a solid and fast routing algorithm is a non-trivial issue. The *Routing over LLN* (ROLL) working group of the *Internet Engineering Task Force* (IETF) proposed the *RPL* routing algorithm as a solution to the routing in LLNs. *RPL* [68] is a

lightweight intra-domain distance-vector routing algorithm for IPv6. It uses *source addressing* (i.e., the sender can decide part or the full path to the destination) by mapping the network topology to a *Direct Acyclic Graph* (DAG) structure composed of one or mode *Destination-Oriented DAG* (DODAG). It supports 6LoWPAN, loop-detection and self-repair.

TinyOS offers a open-source RPL implementation called *TinyRPL* [69].

2.4.3 CoAP

The *Constrained Application Protocol* (CoAP) [28] is a web transfer protocol optimized to work on constrained environments (e.g., WSN) using 6LoWPAN as base and targeting *Machine-To-Machine* (M2M) communications. It implements some basic HTTP requests in a REST-like [29] fashion on top of UDP.

Chapter 3

TinyOS

This Chapter introduces *TinyOS* operating system and how it is commonly used to develop WSN software applications

3.1 Introduction

TinyOS [30] is a framework (i.e., a set of libraries, ready-to-use code and utilities) for developing software application on WSN motes. It provides developers a level of abstraction from the underlying hardware interfaces which allows them to write code without having to focus on the hardware platform details. For this reason, TinyOS is often improperly regarded as a *operating system*[1] is followed.

TinyOS was born as a C language-based software framework, but, after the introduction of the *NesC* language (see Section 3.2), TinyOS has been re-written to adopt such language as the standard programming.

During the evolution of TinyOS, the TinyOS community adopted an *"open proposal"* approach, in which developers can submit documents describing one or more features they wish TinyOS to support. Those documents, called *TinyOS Enhancement Proposals* or **TEP**s, are discussed and, when eventually approved, the features are implemented in TinyOS. TinyOS approved TEPs can be retrieved both via the TinyOS website [30] or simply in any TinyOS code distribution in the `doc/txt/` directory.

Due to the deep architectural changes introduced in the second version of TinyOS (2.x.x), old TinyOS-based applications are not compatible and have to be re-written. To address this and similar issues, one of the main results of this thesis is the proposal of a *software evolution process*, as described in Chapter 15.

[1] Despite the terminology ambiguity, in the following sections, the common approach of referring to TinyOS as an '*operating system*'

3.2 NesC Language in a nutshell

The *Networked Embedded System C* (NesC) is a C-language dialect which add various features and programming paradigms to the C-language, offering to WSN application developers an easy-to-use and powerful programming environment. The NesC language, during the compilation of an application, is trans-compiled into C source code, which is later passed to the cross-compiling toolchain of the target platform for the usual compilation and linking.

The NesC language introduces an *event-based* programming style. WSN software applications are developed as *components*, each one made up by a *configuration* and a *module*: the *module* implements the application logic, while the *configuration* tell the trans-compiler how to connect the component to other components by specifying the so-called *wirings*.

The connection among components is achieved by means of the *interfaces*. A NesC interface contains the definition of a set of *commands* (i.e., the functions callable by means of the interface) and a set of *events* (i.e., asynchronous callback functions).

Every components can *provide* (i.e., implement) interfaces or just *use* them. Components providing an interface have to implement all the commands functionalities, while the events have to be implemented by the components who use the interface.

NesC (and TinyOS) applications can be modeled through the standard UML Component Diagram, in particular, using the "lollipop" (or "ball"). Figure 3.1 shows an example application model.

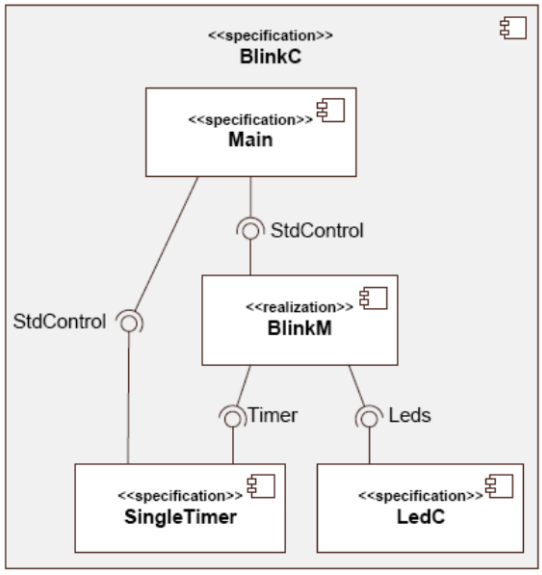


Figure 3.1: UML model of a NesC/TinyOS application

The NesC language provides the so-called *split-phase operations* as support for event-based programming. A split-phase operation is an function invocation which is *splitted* into a command and an event. The command is used to request the start of the operation. This commands almost immediately returns to the caller code (with a return code). Only when the corresponding event is asynchronously called (by the interface provider) the operation is meant to be concluded.

For example, consider the operation of *starting up the RF transceiver*. Suppose an interface called `Radio` with a split-phase operation with:

- the command `Radio.start()`
- the event `Radio.startDone()`

The caller component (i.e., the component which uses the interface) can call the command to start such operation (`Radio.start()`), which returns immediately to the caller. Since the caller component has to implement the events of the interfaces it uses, it has also implemented the `Radio.startDone()` event. This event will be called automatically (asynchronously) when the component providing the `Radio` interface *signals* that the operation has finished. At this point, the caller components can insert (inside the event implementation) the code that need to run after the RF transceiver has been started.

Other NesC features are the following:

- The *task*s, a lightweight multi-tasking primitive which allows to run long and complex operations in the background;
- the *atomic* contexts, used to delimit critical code sections and data so that the access is in mutual-exclusion;
- the *Generic* configurations and modules, which allows to defines one configuration for multiple modules or vice-versa.

TinyOS Active Messages — From version 2.x, TinyOS started using a generic data structure for handle the radio and wired (serial) communications: the *Active Message* (AM). The AMs are defined around the IEEE 802.15.4 frame format and also include *metadata* used by TinyOS to retrieve additional information from the RF transceiver. The AMs support a *send-receive* communication paradigm which is platform-generic. TinyOS provides user applications with a set of interfaces (e.g., `AMSend`) and a set of components (both generic and platform-specific) which can be used or wired to use AMs easily both for radio communications (when the components implement the AM interfaces related to the radio transceiver) or for serial communication (when the components implement the AM interfaces related to a serial communication link).

3.3 TKN154

The communication layer in TinyOS is composed of only a set of abstractions which aims to provide a simple frame-level *send-receive* paradigm, which is what

most applications requires. Developers who require a full IEEE 802.15.4 MAC layer have few alternatives apart from manually writing a MAC layer themselves.

A first open-source IEEE 802.15.4 MAC layer to be used in WSN motes was OpenZB [67]. At the same time, other research groups were working on providing a hardware-independent MAC layer. The combined efforts resulted in the **TKN154** [31] MAC layer. The TKN154 is a MAC layer implementation for WSN motes and for TinyOS-based applications. It is written in NesC and it provides the core components for IEEE 802.15.4 compliant communications.

The TKN154 focuses on *platform independence*, *modularity* and *extensibility*. It provides the core services of the MAC layer using the available PHY layer provided by TinyOS and the RF transceiver drivers. The TKN154 is organized into different abstraction levels, from the low-level RF transceiver communication components to the high-level wrappers components (for both beacon-enabled and beaconless communications) which provide the standard MAC SAP and its interfaces (e.g., `MLME-ASSOCIATE`). Figure 3.2 shows an overview of the TKN154's architecture.

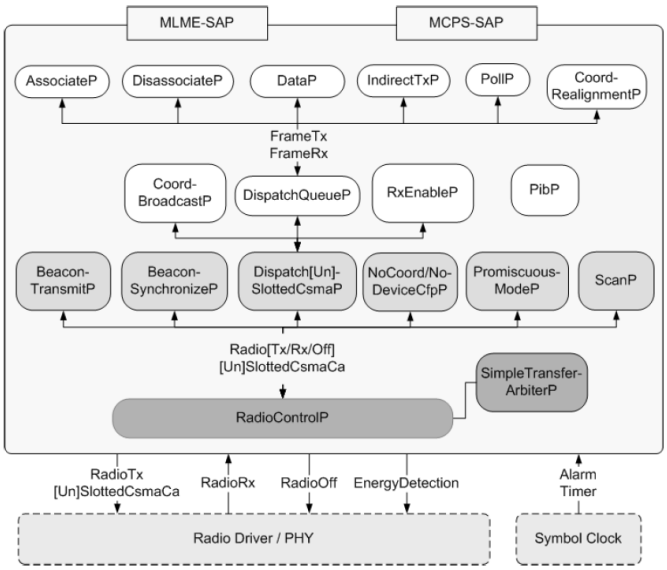


Figure 3.2: TKN154 architecture

As highlighted by [31], tests performed on common WSN mote platforms equipped with the *CC2420* shown that there are timing issues (drifts) which causes WSN motes to lose their synchronization. These issues have been recently solved by the radio chip manufacturer, with the release of a new version (*CC2520*) which, however, still not replaces the old version in the designs of the WSN platforms mentioned in Section 2.1.2 at the time of writing.

Chapter 4

Middlewares for WSN

This Chapter briefly report on the state-of-art of the middleware for WSN, focusing then on a particular class of middleware, called *Mobile-Agent Middleware*, which was adopted for our research activities.

4.1 Introduction

Although TinyOS (or any other solution) offers a solid hardware abstraction layer, often the complexity of the final application requires an additional level of abstraction to e.g., ease the development or to provide context-specific features. In these situations, a *Middleware* (MW) can be adopted. As the name suggests, a MW is an intermediate software layer which lies between the low-level software layer (e.g., the operating system) and the final application.

In general, a MW provide context-specific features and/or an application-oriented software environment. In the context of WSN, using a MW can greatly enhance the application development experience.

WSN MW should guarantee the following extra-functional requirements [40]:

- *Reliability.* WSN are vulnerable to node failures so, a robust MW should be able to overcome such failures without interrupting the WSN services. This requires that WSN should implement appropriate recovery strategies.
- *Re-configurability.* MW must be able to handle ever changing networks, both in term of number of nodes and network topology/architecture. Ideally, the mote reconfiguration should be provided without interrupting the WSN service.
- *Heterogeneity.* A MW should provide an abstract interface whatever kind of nodes composes the WSN, since different hardware nodes from different technologies could be present in the network.
- *Battery life.* Energy management is always very relevant in MW for WSNs. To guarantee long life for the battery, MW should provide an effective use

of the energy-aware communication protocols, *Smart HW handling* (i.e., to power off unused HW components) and the *Data Aggregation* support (i.e., to select or aggregate data, reducing as much as possible transmissions to the sink node).

- *QoS.* A MW should help QoS management by measuring performance, network capabilities, throughput, power-consumption and transmission delays.

- *Real-Time requirements.* When WSN applications need real-time data, a MW should provide real-time services in spite of limited node computational power and HW resources.

- *Context-Awareness.* A MW should be able to adapt itself to surrounding environment, composed by HW/SW resources, physical characteristics and constrains, that are not under the control of the WSN, that the MW should be aware of.

- *Security.* Since WSN are developed to be deployed also into sensible environments, in some WSN applications, security is even more important than data. For this, a MW should be capable of providing security-centric mechanisms, granting data integrity, authentication techniques, and secure transmissions. However, common techniques are not always applicable, since there could be performance issues and/or unacceptable power consumption. So, they are often simply ignored. In other cases (e.g., [18]), such mechanisms are customized to reduce the performance drawback while being still quite effective.

Trying to take into account previously listed design issues, WSN MW technologies are in continuous progress. So, while it is not possible to describe details of specific products, it is possible to identify a common classification. The reference architectures for WSN MW are:

- *Data-base MWs.* MWs of this kind view the WSN itself as a distributed database. Data retrieval is performed by means of query-based abstractions. A famous example of this category is *TinyDB* [38].

- *Virtual Machine-based MWs.* This kind of MWs uses a virtual machine approach, providing a flexible environment and an interpreter. An example is reported in [41].

- *Application-driven MWs.* As the name suggests, this kind of MWs are driven by the final application requirements, which define how the network should operate. An example is *Milan* [42].

- *Mobile-Agent MW* (MAMW). An *Agent* is an object, composed of code and some supporting data structure, that can move (migrate) among different nodes of the network. Such MWs provide the features of VM-based MWs enriched with high-level primitives to control migrations.

An MAMW allows to achieve better resilience (by adding code and data redundancy) and scalability (by offering a dynamic way to remotely program new devices). Moreover, it is possible to *re-program* WSN nodes on-the-fly while keep them operative.

4.2 Agilla

Agilla [34] is an MAMW for WSNs based on TinyOS. It allows the agents to be created, substituted and destroyed at run-time, without stopping the execution of the code. The agents are written in a assembly-like language that is interpreted by the corresponding virtual machine. Agilla middleware allows to use a new paradigm for programming and using sensor networks, where applications consist of special programs called mobile agents that can migrate their code and state from one node to another as they execute. Mobile agents offer an unprecedented level of flexibility by allowing applications to spread throughout the network and to position themselves in the optimal location for performing their task.

Agilla guarantees a high degree of reconfigurability (i.e., agents can be injected, moved, cloned, replaced and every physical node can run multiple agents at a time), reliability (i.e., if an agent crashes, it does not affect the functionality of the hosting WSN node nor the other agents running on it). Also, since the deployment of the agents (i.e., the application) is dynamic, there is the possibility to dynamically create and distribute agents to fit the specific context requirements.

Figure 4.1 shows an overview of Agilla's architecture.

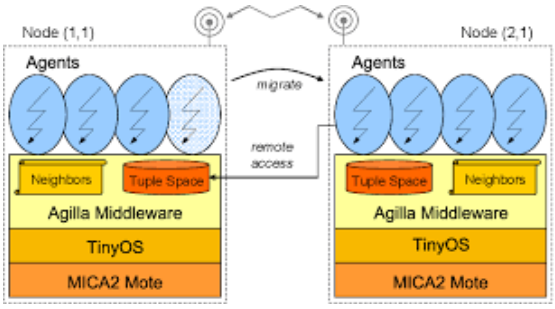


Figure 4.1: Agilla MW architecture

4.2.1 Agilla agents

Agilla agents are small pieces of assembly-like code which are *compiled* into a bytecode before being sent and loaded in one or more motes of the WSN (i.e., agent *injection*). The agent, then, is executed (i.e., interpreted) by the

Agilla Engine component. The Agilla agent *Instruction Set Architecture* can be consulted in [43] while Figure 4.2 shows an example agent.

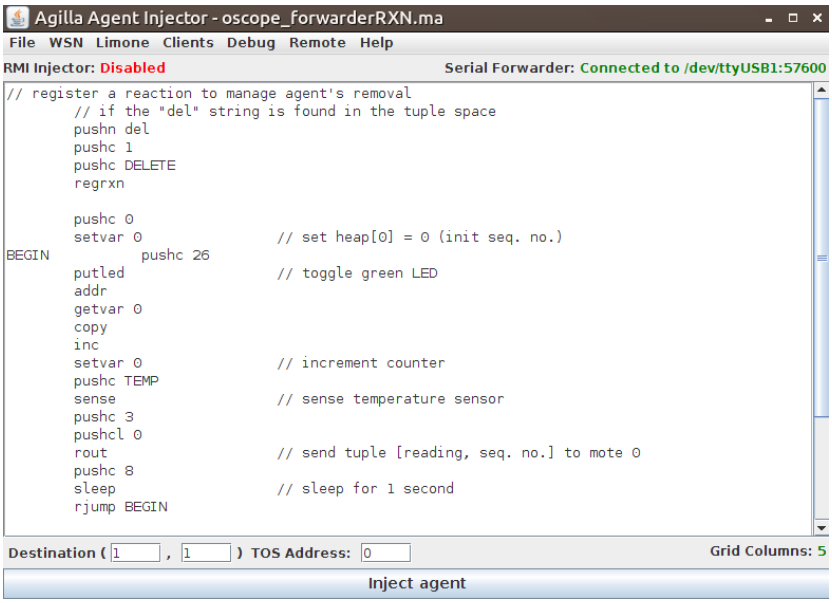


Figure 4.2: Agilla *Agent Injector* with the code of an agent

Multiple agents can run at the same time on a single mote. Each agent has a *operand stack* which is used by the Agilla ISA as temporary storage for operands. The *Heap* is instead used as a more permanent storage for data. A different data structure is instead the so-called *tuplespace*. The tuplespace (of a node) is shared by all the agents running on the same mote and can store *tuples*, which are key-value pairs.

Agents can register *reactions* by specifying a code label and a tuple *template* to be matched against all the tuple in the tuplespace of a mote. When a tuple match the template, the agent execution is immediately moved to the label in a *interrupt-like* behavior.

4.2.2 Agilla and TinyOS 2.x

Since the transition of TinyOS from version 1.x to 2.x and the lack of compatibility between the two versions, Agilla cannot be compiled successfully against newer versions of TinyOS and thus it cannot gain the introduced advantages and the support for new platforms and protocols. Chapter 15 describes the research activities addressing this problem.

Chapter 5

WSN Protocols Security

This chapter introduces the security primitives which the IEEE 802.15.4 provides, the drawbacks and what is missing to secure out WSN communications. In particular, first section describes what the IEEE 802.15.4 baseline is, then a new standard (the *IEEE 802.15.9* aimed to solve the key-transport problem is described and, finally, the chapter closes with a discussion on the security measures adopted in higher layers (e.g., ZigBee).

5.1 IEEE 802.15.4 Security

The IEEE 802.15.4 standard describes the expected security-related features in term of CIA triad (*confidentiality*, *integrity* and *authentication*).

In particular, the standard describes:

- optional frame header fields for enabling the security features;
- a set of parameters used to decide which level of security to adopt;
- the symmetric cryptographic algorithm to be used and its mode of operation;

5.1.1 Security-related Header Fields

In the first field of the MAC frame header, the *Frame Control Field*, the *Security Enable* bit tells whether the receiver of a frame should expect the presence of additional security-related fields or not. If so, the *Auxiliary Security Header* (ASH, Figure 5.1 is the first and most important field about the security aspects of IEEE 802.15.4.

The ASH has a variable length, from 1 to 14 bytes. Only the first byte is mandatory and contains the *Security Control Field*. This field contains the selected security level (in term of key length and authentication), how the key is specified (explicitly or implicitly e.g., via an index) and how long are the following fields if present.

Octets: 1	0/4	0/1/5/9
Security Control	Frame Counter	Key Identifier

Figure 5.1: IEEE 802.15.4 Auxiliary Security Header

Following the first byte, there is the optional *Frame Counter*, which, if present, it is used to generate the *nonce* used in the symmetric cipher and for general replay protection.

The last optional field is the *Key Identifier*. Its content depends on the Security Control field and it contains the *originator* of the key and an index to the key to use.

5.1.2 Symmetric Cipher

The standard uses the *Advanced Encryption Standard* (AES, Rijndael) [44] symmetric block cipher with a fixed key length of 128 bits (16 bytes). The mode of operation to be used is a variant of the *Cipher Block Chaining with Counter Mode* (CCM) called *CCM**. This mode of operation provides both encryption and an variable-length *authentication tag* obtained via the CBC-MAC [81]. Additional information on the CCM* mode can be found in the appendices of the standard ([16]).

Unfortunately, the standards lacks of the description on *how* the symmetric keys are deployed, exchanged and updated dynamically in the WPAN. A partial solution to this issue has been introduced by a further standard, the *IEEE 802.15.9*.

5.2 IEEE 802.15.9 Standard

The IEEE 802.15.9 standard [17] describes the recommended practices for implementing a IEEE 802.15.4-compliant *Transport and Key Management Protocol*. It introduces a new layer on top of the IEEE 802.15.4 MAC, the *MPX* which is meant to adapt an existing key management protocol (KMP) to be used to generate and transport cryptographic keys. The IEEE 802.15.9-based protocol stack is shown in Figure 5.2.

The MPX layer provides also additional features:

- Support to *fragmentation* and *re-assembly*. The standard adds the possibility to break up a single message into multiple *fragments* which are collected and re-assembled by the receiver. This allows to transport long cryptographic material with a single logical transmission.

- Support for the *Multiplexing* of communications. An additional field in the IEs is added to support different upper-layer protocols.

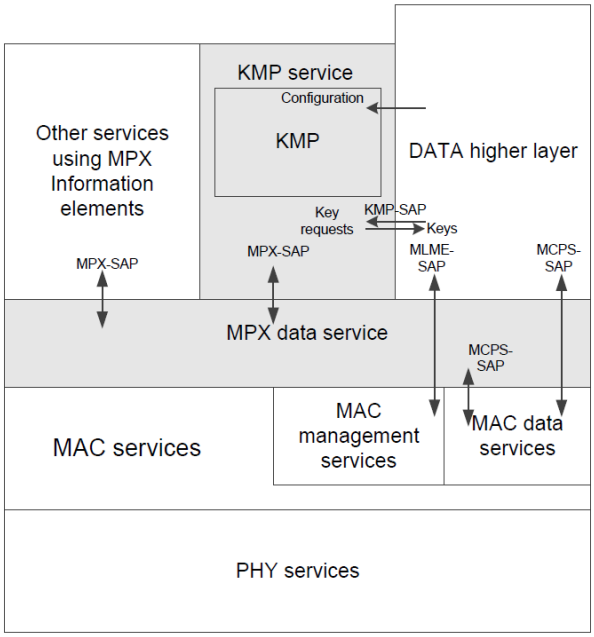


Figure 1—System overview

Figure 5.2: IEEE 802.15.9-based protocol stack

The KMP service is the adapter layer used to connect the MPX layer with the real KMP. This service provides the upper layers with a SAP for generating, changing or deleting keys via the inner KMP.

Although in the standard, additional information about some already supported KMP are present (in the appendices), the presented research activity sets the bases for a new KMP service featuring a hybrid cryptography scheme as inner KMP, as discussed in Section 10.6.

5.3 ZigBee Security

Security in ZigBee is based on the security primitives already defined in the IEEE 802.15.4 standard and add additional security-related features in the NWK layer to improve and manage lower-level security mechanisms. ZigBee provides three basic security features:

- Encryption/Decryption
- Anti replay-attacks mechanisms
- Access control lists

Normally, ZigBee security is *distributed*, but, if required, it can be *centralized* into a single entity (which could be the WPAN coordinator) called *Trust Center* which is responsible for authenticating the joining nodes and distributing the keys.

Encryption is based, as in IEEE 802.15.4, on the AES cipher with 128bit keys and CCM* mode. There are different sets of keys:

- The *Network Key* (NK). This key is shared among all the network nodes and it is used for broadcast communications. The NK is also used to protect node communications when *joining* the network.
- The *Link Key* (LK). This key is used for point-to-point communications. Depending on the security mode (centralized or distributed), the LK can be a *global key* (used by the Trust Center), a *TC Link key* (used by the Trust Center and a single node) or a *Application Link Key*.

Figure 5.3 shows the summary of the ZigBee cryptographic key types.

Security Key	Description	
Network-level Security		
Network key	• Essential key used to encrypt communications between all nodes of the network • Randomly generated by the Trust Centre • Distributed to joining nodes, encrypted with a pre-configured link key (see below)	
Application-level Security		
Global link key (pre-configured)	• Used between the Trust Centre and all other nodes • Pre-configured in all nodes (unless a unique link key is pre-configured - see below) • Also used in joining to encrypt network key transported from Trust Centre to joining node • If ZigBee-defined, allows nodes from all manufacturers to join the network • If manufacturer-defined, allows only nodes from one manufacturer to join the network • Touchlink Pre-configured Link Key is a key of this type • Distributed Security Global Link Key is a key of this type	
Unique link key	Optional key used to encrypt communications between a pair of nodes - may be one of:	
	Pre-configured unique link key	• Used between the Trust Centre and one other node • Pre-configured in Trust Centre and relevant node • Also used in joining to encrypt network key transported from Trust Centre to joining node • Install Code-derived Pre-configured Link Key is a key of this type
	Trust Centre Link Key (TCLK)	• Used between the Trust Centre and one other node • Randomly generated by the Trust Centre • Distributed to node encrypted with network key and pre-configured link key (if any) • Replaces pre-configured link key (if any) but application must retain the pre-configured key in case it needs to be reinstated
	Application link key	• Used between a pair of nodes, not including the Trust Centre • Randomly generated by the Trust Centre • Distributed to each node encrypted with network key and pre-configured link key (if any)

Figure 5.3: ZigBee Cryptographic Keys

Depending on the ZigBee protocol configuration, it is possible to adopt a

Certificate-based key establishment: every node stores its certificate (which has to be issued by a certification authority) which is sent/verified to establish a shared key. For this step, ZigBee adopts the $ECMQV$ protocol [76].

Additional information on ZigBee security and Key establish mechanisms can be found in [74] and in [75].

Chapter 6

Cryptography for WSN

This chapter describes the cryptography-related techniques and software tools adopted in the context of WSN. As first step, the requirements, constraints and the basic cryptography protocols involved are described; then some of the solutions adopted to overcome to the WSN limitations are presented.

6.1 Overview

Although it is not its only application, *Cryptography* is widely used as mean to protect communications from unauthorized parties. In the context of WSN, cryptography is even more important, since WSN nodes communicate using a channel (electromagnetic waves) which are easily accessible to everyone who is equipped with a compatible radio apparatus.

Cryptography protect communications by providing solutions which fulfill, among the other requirements, the so-called *CIA Triad*. The CIA triad is represented by the following three requirements:

- The *Confidentiality* requirement states that communications sent to a party should be accessible only to it and no other party.
- The *Integrity* requirement states that there should be possible to determine whether a message has been modified prior to be received by the destination.
- The *Authentication* requirement states that should be possible for a receiver to determine which is the source of the communication (i.e., the sender). Authentication is also linked to the *Non-Repudiation* of a communication, i.e., a sender, once authenticated, should not be able to repudiate a message it sent.

In addition to these requirements, depending on the context, other requirements can be considered:

- The *Availability* requirement, i.e., the parties should be operational at any given time, despite any workload or any malicious attempt to slow down or stop their operations (e.g., *Denial of Services attacks*).
- The *Resistance to Tampers*, i.e., the parties should be resistant to physical tampering activities (e.g., node sabotage) and to data tampering activities (e.g., node reprogramming).
- The *Ability to detect network threats*, i.e., the parties should be able to detect incoming or in-progress attacks and attackers to possibly react and/or to send notifications to network operators.

The following sections provide a description of the state-of-the-art solutions used to fulfill the above requirements.

6.2 Symmetric Cryptography

In order to fulfill the *Confidentiality* requirement, one core component of any cryptography-related solution is the *Symmetric Cryptography*. As the name suggests, in symmetric cryptography ciphers the key used to encrypt and decrypt data is the same for all the parties involved in a given communication. Symmetric schemes algorithms are usually fast and lightweight, with the only drawback in the fact that the (shared) key has to be distributed before any encrypted communication.

6.2.1 Block Ciphers

Block Ciphers are, as the name states, symmetric-key ciphers which operate on the plaintext by splitting it intro *blocks* of a fixed length. The cipher takes the key and an input block of plaintext and produces a output block of the same size. Common block sizes are 8, 16 or 32 bytes.

If the plaintext length is not divisible by the block size, a *padding* scheme has to be adopted. A common padding scheme is the one described in the *Public Key Cryptography Standard N#7* (PKCS#7) [50] which consists in, given the length of the plaintext l and the required block size k, appending $k-(l \mod k)$ bytes to the plaintext with a value of k. For example, consider the following stream of bytes as plaintext *0x11 0x22 0x33 0x44 0x55 0x66* i.e., $l = 6$ and a block size of $k = 16$. Since $16-(6 \mod 16) = 10 = 0x0A$, the padded plaintext will be `0x11 0x22 0x33 0x44 0x55 0x66 0x0A 0x0A 0x0A 0x0A 0x0A 0x0A 0x0A 0x0A 0x0A 0x0A`

6.2.2 Operating Modes

When processing a plaintext composed by multiple blocks, block ciphers can use different *operating modes*. The trivial operating mode, called *Electronic Code Book* (ECB), consists in processing each input block separately and then

concatenating each output block. This operating mode, however, makes the block cipher *vulnerable* to different attacks, e.g., *block-reuse attack* and *replay attacks*.

In order to overcome the ECB mode issues, various other modes can be used:

- *Cipher Block Chaining* (CBC): each input block is *XORed* with the previous output block. The first block is instead XORed with a special value called *Initialization Vector* (IV). CBC mode is vulnerable to *padding oracle attacks* i.e., attacks which attacks the padding scheme to infer, one byte after the other, the plaintext.
- *Cipher FeedBack* (CFB): each input block is the output block of the previous iteration. Each output is the result of the cipher XORed by the corresponding block of plaintext. The first input is an IV. The *Output FeedBack* mode is similar, with the only difference being that input blocks are directly the result of the cipher (before being XORed with the plaintext).
- *Counter* (CTR): each input block is created by the combination of a *nonce* and the value of a counter. The result of the block cipher is XORed with the plaintext and the counter is incremented. Since CTR mode is fast and lightweight, it is often used in WSNs and in other performance-sensible platforms,

6.2.3 *Authenticated Encryption*

In addition to the operating modes mentioned above, some *Authenticated Modes* has been introduced. These modes of operation generate also an *authentication tag* which can be used to perform plaintext/message authentication:

- *Counter with CBC-MAC* (CCM) uses the CTR mode for encryption and CBC mode for producing the authorization tag. The latter is the result of the last processed block in CBC mode (using an IV of `0x00` bytes). This mode is vulnerable if the key used to encrypt in CTR mode is the same as the key used to compute the CBC-MAC. A variant of the CCM mode is also used in the IEEE 802.15.4 standard.
- *Galois Counter Mode* (GCM) uses the CTR mode for encryption and the *GHASH* function for computing the authentication tag. More information can be found in [45].

An example of well-known and reliable block cipher is the *Advanced Encryption Standard* (AES), which uses 16-bytes blocks and it is commonly used with CTR, CBC, CCM or GCM.

6.2.4 Stream Ciphers

The *stream ciphers* are symmetric-key ciphers that process *streams of bytes* instead of blocks. To do so, a infinite *key stream* is created from the initial key

with a *Pseudo Random Number Generator* (PRNG). They are commonly used in scenarios where the length of the input plaintext is often unknown.

A recent, famous and reliable stream cipher is the *Salsa20/ChaCha20*, a family of stream ciphers [46].

6.3 Public-Key Cryptography

In the *Asymmetric Cryptography*, commonly referred as *Public-Key cryptography*, a set of two keys has to be used for cryptographic operations. Of the two keys, one is called *Private* key and is meant to be kept secret by its owner. The other is called *Public* key and should be available to every party.

Public-key cryptography solves the key distribution issue of symmetric cryptography. In particular, it is common to use Public-key cryptography to safely exchange (or, better, *agree on*) a symmetric key (e.g., using the Diffie-Hellman protocol, [47]).

Public-keys are commonly distributed by means of *Digital Certificates*. These certificates are used to prove the ownership of a public key, hence, they provide both authorization and non-repudiation of messages encrypted using it. Digital Certificates are issued by the so-called *Certificate authority* (CA).

The set of all the roles, platforms and procedures associated with Public-Key cryptography and the distribution of certificates is commonly referred as *Public Key Infrastructure* (PKI).

6.3.1 Protocols

In the case of *Public Key Encryption*, the sender (e.g., *Alice*) uses the receiver's (e.g., *Bob*) public key to encrypt the message. In this way, only who possesses the corresponding private key (i.e., Bob) can successfully decrypt the message

In contrast, if Alice wants to append a *Digital Signature* to a message, she uses hers private key to forge a *Public Key Signature* from the message. When Bob receive such message, he can *verify* the signature by using Alice's public key. If the verification is successful, Bob knows that the message has Alice as sender and, at the same time, Alice (whose public key can be checked by Bob and the CA) cannot repudiate the message sent.

Different classes of Public-Key cryptography schemes exist. In the following sub-sections, the two main classes are considered: the *factorization* based ciphers and the *discrete logarithm* based ciphers.

6.3.2 Factorization-based Ciphers

The foundations of the *Factorization-based* ciphers is in the *factorization* problem. Given a very large number n which is product of a set of large prime numbers $p_1, p_2, ..., p_k$, it is easy to compute n given the primes (by a simple multiplication) but it is *very* expensive to find the primes from just the value

of n. In fact, in the general scenario, the only way to find the primes is actually factorize n, a problem which has no efficient solution.

The best-known factorization-based Public-Key cipher is *RSA* [48].

6.3.3 Discrete Logarithm-based Ciphers

Given the set of integers modulo a large prime number p, a *base* a and an integer b, the *Discrete Logarithm* problem (DLP) is to find x such that:

$$b = a^x \mod p$$

Computing the value of b is easy if x is known, but, without such knowledge, there are no trivial solution for the general case apart from using *brute force* techniques. In some special cases (e.g., when using a sub-group with an order which is a *smooth* integer), the computation speed can be enhanced using special algorithms.

The *Discrete Logarithm Problem over Elliptic Curves* (ECDLP) is a similar problem but has a different construction. It is used in the elliptic curve -based Public-Key ciphers, which is introduced in Chapter 7.

One of the most famous DLP-based cipher is *ElGamal* [49].

6.4 Hybrid Cryptography

Apart from Symmetric and Public-Key cryptography, it is possible to adopt *mixed* approaches (*Hybrid* Cryptography). In particular, these approaches are commonly adopted to overcome specific limitation or increase the flexibility of the schemes.

Hybrid Cryptography schemes are very different one from the other. This thesis focuses on Hybrid Cryptography schemes based on *vector algebra*. Two of these schemes are described in Chapter 10 and in Chapter 11.

6.5 Cryptography-related Topics

6.5.1 Secure Random Numbers Generation

One often overlooked problem in cryptography is the *secure* generation of random numbers. This topic is very important, since the security level of most of the cryptography schemes depends directly the real randomness of the generated random values. Generating *real* random numbers is not possible for any *deterministic* machine, so in practice, *Pseudo* random numbers are generated. Although they are not *as random as random numbers*, sequences of pseudo-random numbers are constructed so that is unfeasible for anyone to *replicate* them. An example of a pseudo-random number, is the number of current count of clock cycles of a processor at a given time. This number is difficult to guess but, it is not random since it is possible, with proper devices or additional information, to find it.

Although there are various methods and algorithms for generating pseudo-random numbers, in cryptography often this is not enough. In fact, cryptography relies on *Cryptographically-Secure* random number generators (CSPRNG), which have the mathematical assurance that the number generated are unfeasible to replicate. To do so, such generators needs a *seed* (i.e., a value used to initialize the generator) which must come from a source with **Very High** entropy. In computers, such entropy comes often by mixing different source (e.g., time, keyboards, mouse movements, sensor reading etc.). An example of CSPRNG is the *Blum-Blum-Shub* algorithm [33], which is formally proven to generate secure pseudo-random numbers with a proper source of entropy.

6.5.2 Hash Functions

A *Hash* (called also *Message Digest* or just *Digest*) is a fixed-sized value which *represent* an arbitrary-length data. Hashes are created from specific functions called *Hash Functions*. Such functions have the following properties:

1. They are *trap-door* functions, i.e., it is easy to compute a hash from a message but is unfeasible to find a message which gives a target hash.

2. They are resistant to *collisions*, i.e., it is difficult (although not impossible) to find two or more different messages on which the hash function returns the same hash value.

Hash functions applications are various, although the two main contexts are *Hash Functions* for *Hash-based* data structures (e.g., Hash tables) and *cryptographic hashes*. The former focuses on generating a small hash value (e.g., 32 or 64 bits) and collisions are tolerated and/or avoided using different techniques (e.g., linear hashing, double hashing etc.). The latter, instead, have no hard restrictions on the size (224/256/512 bits are common sizes) but collisions are absolutely to avoid, since they could undermine the security of the function itself.

In cryptographic protocols and, in particular, in digital signature protocols, cryptographic hash functions are adopted to create a fixed-size equivalent of the message to sign so that it is usable (and verifiable) no matter the original size.

Common secure cryptographic hash functions which are still considered secure (i.e., no methodology to find collisions is known) are the *SHA 2 suite* (e.g., *SHA-256*) [51] and the recent *SHA-3* (Kekkak) [52].

6.5.3 Message Authentication

Message Authentication Codes (MAC), also called *Keyed-hashes* are fixed-size values that are similar to hashes but also are bound to a particular cryptographic key i.e., the MAC of a message computed with a key K_A is *different* from the MAC of the same message computed with a key $K_B \neq K_A$. The functions used to compute MACs (*MAC Functions*) are usually built on secure cryptographic hash functions or on symmetric ciphers. A good MAC function

is easy to compute if both the message and the key are known. If the key is missing, it is difficult to compute the same MAC from a message even with the message available.

As an additional property, a MAC computed on a message using a shared-key by a party (e.g., *Alice*), *binds* the MAC computation to the party itself, so that any other party (e.g., *Bob*) receiving both a message and such MAC can recompute it (assuming he knows the shared key) and compare the computed MAC with the one received. If they are equal, *Bob* can be sure that was *Alice* (or anyone knowing the shared key) to send the message.

As the name suggests, MACs and MAC functions are used to provide *authentication* similarly to digital signatures but without the public-key mechanism. Every secure communication protocols usually uses encryption and a MAC to both achieve confidentiality and authentication. In this sense, two approaches are usually adopted:

- Encrypt-Then-MAC: the message is encrypted and the MAC is computed on the *ciphertext* and left un-encrypted.
- MAC-Then-Encrypt: the MAC is computed on the plaintext and then encrypted along with the message itself.

Typical MAC functions are the $HMAC$[51] which is defined upon an hash function $H(m)$:

$$HMAC(M, K) = H((K' \oplus opad)||H((K' \oplus ipad)||M'))$$

where M' and K' are, respectively the message and the key padded to the required bit length while $ipad = 0x363636...36$ and $opad = 0x5C5C5C...5C$ are special padding values.

Another common MAC function is the $CBCMAC$[81] which is computed from a symmetric block cipher used in CBC mode and taking its last output as MAC. A notable example is the AES-128bit-CBCMAC.

As final note, notice that is not recommended to use the same key both for encryption and MAC computation, since it can lead to well-known attacks (e.g., [82]

6.5.4 Message Integrity

The *Integrity* requirement is usually fulfilled by inserting in a message an *integrity code*. This integrity code is computed using *Error Correction Codes* generation, which allow both to detect integrity violations and, in some cases, correct them. In the context of security, those codes can detect when messages are tampered by an attacker intentionally to disturb the communication.

Despite hash functions could be used to perform a similar task, error-correcting codes are usually smaller and easier to compute.

A basic example of integrity code is the *parity bits* i.e., an additional bit is appended to data to indicate whether the number of 1s (or 0s) is *odd* or *even*.

A little more complete approach is to compute a *checksum* (i.e., summing up all the bytes values in the messages) or, even better, using a *Cyclic redundant code* (CRC)[26] which is a special class of error-correcting codes which adopts a checksum computed using Galois field arithmetics.

Chapter 7

Elliptic Curve Cryptography

This Chapter introduces the *Elliptic Curve Cryptography*, the operations defined on a *Elliptic Curve* (EC) and the cryptographic protocols used to provide confidentiality and authentication.

7.1 Overview

The *Elliptic Curve Cryptography* (*ECC*) is a set of cryptographic techniques introduced in 1985 by Miller and Kobliz [53]. In the recent years, it has been rediscovered and adopted as mean to provide public-key cryptographic schemes. ECC is based on operations and existing relations found in *elliptic curves* defined over *finite fields* and their *points*. ECC, when used in the context of cryptographic protocols, provide mechanisms for encryption, digital signatures, key exchange, random number generation and in integer factorization algorithms.

The main advantage of ECC is the reduced size of the cryptographic keys in comparison with other public-key protocols. Table 7.1[70] shows a comparison of the size of keys (in bits) in respect of the security level.

	public key length	equivalent symmetric key length	Subjective Security
RSA	1024	80	Not Recommended
	2048	112	Good
	3072	128	Suite B TOP SECRET
	15360	256	Future
ECC	163	80	Not Recommended
	224	112	Good
	256	128	Great
	384	192	Suite B TOP SECRET
	521	256	Future

Figure 7.1: ECC security level comparison with RSA

7.2 Curves

This section analyze the required mathematical background needed to construct the ECC.

An elliptic curve is the set of points (x, y) which are solution of:

$$y^2 = x^3 + ax + b \tag{7.1}$$

or of:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{7.2}$$

. An elliptic curve can be defined in various numerical field. Equation 7.1 can be used in every field of characteristic different from 2.

In ECC, the common numerical fields adopted are:

- Prime-based fields (or *prime fields, prime curves*, etc.), which are the integers modulo a prime number p, i.e., $\mathbb{Z}/\mathbb{Z}_p$;
- Galois finite fields (or *binary fields, binary curves*) which are the polynomials with characteristic 2 modulo an irreducible polynomial.

In Figure 7.2 some example curves are shown (in $\mathbb{R}^2$).

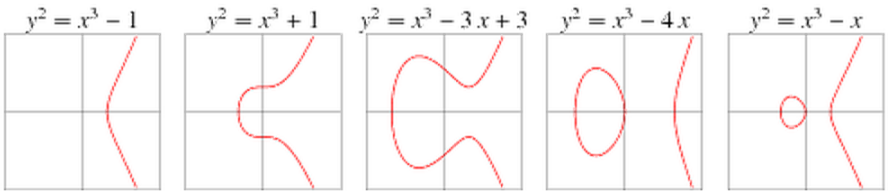


Figure 7.2: Examples of elliptic curves

The effective number of proper points in a prime/binary curve is not directly computable, the only approximation (i.e., a limitation) is given by the famous Hasse Theorem [71]:

$$|N - (q + 1)| \leq 2\sqrt{q}$$

In order to clearly define the set of available points, a *cyclic sub-group* is defined on the curve. To do so, a special point, called *Generator* (**G**) is chosen for a given curve. This point has the property that, by multiplying it (see Section 7.4) by $n + 1$, the result is G itself. The number n is another curve parameter which is defined once G is chosen and represents the number of different points which can be obtained as multiple of G. The selection of G is performed to obtain the best mathematical properties and the larger possible value for n.

In addition to the proper points on the curve, the infinity point $\mathcal{O}$ is added to the set. This point represents the neutral value for addition (i.e., every point added to $\mathcal{O}$ results in the point itself). Using the affine coordinates (carthesian), $\mathcal{O}$ cannot be represented directly but it is assumed to be at $(0, +\infty)$ and at

$(0, -\infty)$. Using other kind of coordinate systems (e.g., *projective coordinates*) the point $\mathcal{O}$ is instead represented in a closed form (e.g., $(\cdot, \cdot, 0)$).

Depending on the field used to define the curve, a set of parameters and operations are defined on the curve and on the curve points. The sections below describe the *prime-fields* and the *binary-fields* cases and the basic arithmetics used in ECC in those fields

7.2.1 Prime-fields Curves

A curve defined on a prime-fields has the following *Domain parameters*:

$$\mathcal{D} = \{a, b, p, G, n, h\}$$

where a and b are the constants used in the formula of the curve (Equation 7.1), p is the prime number which defines the field, G is the generator of the cyclic sub-group, n is the *order* such that $nG = \infty$ and h is the *co-factor* and it is an integer which represents the ratio of the total number of points in the curve and the number of points in the sub-group.

7.2.2 Binary-fields Curves

In the case of binary-fields, the curve is described by Equation 7.2 with the following parameters:

$$\mathcal{D} = \{m, f, a, G, n, h\}$$

where m and f define the irreducible polynomial used to define the Galois Field ($F(x) = x^m + f_{m-1}x^{m-1} + ... + f_0$), while a stands for the coefficients $a_n...a_0$ used in the curve definition. G, n and h keep the same meaning also in binary-fields curves.

7.3 Field Operations

Before describing the core operations on elliptic curves, this section recaps on the basic (modular) arithmetics involved.

7.3.1 Modular Additions, Subtractions and Multiplications

All the basic operations are performed in modular arithmetics. In the case of prime-fields, the set involved is $\mathbb{Z}/\mathbb{Z}_p$ with p prime number (i.e., the set of positive integers less than p). Addition and subtraction results are reduced modulo p (see next section) but, in the case of multiplications, often additional techniques are used (e.g., the *Karatsuba* multiplication [72]) to improve performances.

When binary-fields are involved, the arithmetic operations are performed modulo an irreducible polynomial $F(x)$. Polynomials can be represented in sequences of bits and addition and subtraction are the same operation and it

is performed via the XOR binary operation. Multiplications can be performed with a series of XOR-and-Shift operations. For example:

$$F(x) = x^8 + x^4 + x^3 + x + 1$$

$$a = x^3 + 1$$

$$b = x^6 + x$$

$$ab = x^9 + x^6 + x^4 + x$$

$$ab \mod F(x) \equiv x^6 + x^5 + x^2$$

in binary representation:

$$F = 0100011011$$

$$a = 0000001001$$

$$b = 0001000010$$

$$ab = a * x + a * x6 = (a << 1) \oplus (a << 6) =$$

$$0000010010 \oplus 1001000000 = 1001010010$$

$$ab \mod F \equiv 0001100100$$

7.3.2 Modular Reduction

The modular reduction is one of the most important field operation. It computes, given an integer a and a prime number p, the *remainder* of the division of a over p. Basic techniques are the classical division or repeated subtractions. When the size of the operands involved is big (e.g., hundreds of bits), these techniques are performance bottlenecks. In prime-fields, to overcome this issue, specific techniques can be used, e.g., the *Montgomery Reduction* [60]. This technique involves applying a transformation from the original adopted field (e.g., $\mathbb{Z}/\mathbb{Z}_p$) into a more convenient $\mathbb{Z}/\mathbb{Z}_R$ field with $R = 2^n$ and such that $R > p$. Using $\mathbb{Z}/\mathbb{Z}_R$, it is possible to perform modular reductions with simple and extremely efficient *AND-masking* operations, while divisions and multiplications can be performed with binary shifts. Once the modular reductions have been performed, the inverse transformation can be applied to return to the original modular field.

For example, consider the field $\mathbb{Z}/\mathbb{Z}_{13}$, i.e., the set of all the positive integers modulo $N = 13$. Normally, modular reductions would involve at least one division by 13 (considering the value to be transformed be $T = 10$). Using Montgomery transformation, we choose $k = 5$ and $R = 2^k = 32$. We start by

computing the value N' such that $13 * N' \equiv -1 \mod 32$, which, in this case results in $N' = 27 \mod 32$. Then, we can retrieve the transformed value t:

$$m = ((T \mod R) * N') \mod R = (10 + 27) \mod 32 \equiv 14 \mod 32$$

$$t = (T + Nm)/R = (10 + 13 * 14)/32 = 192/32 = 6$$

The anti-transformation consists in a simple division by R (i.e., a multiplication with the modular inverse of $R \mod N$):

$$R'R \mod N \rightarrow R' \equiv 11 \mod 13$$

$$t * R' \mod N = 6 * 11 \mod 13 \equiv 10 \mod 13$$

.

An application of this transformation is to efficiently compute products by transforming the operands, computing the product and anti-transforming.

7.3.3 Modular Inversion

Modular inversion is the most complex core arithmetic operation in modular fields. This operation is equivalent to the division for integers, i.e., to find a value x^{-1} such that $x * x^{-1} = 1 \mod p$. In general modular fields, this operation has a guaranteed result only if p is prime, while in binary fields, $F(x)$ has to be an irreducible polynomial of the field i.e., the polynomial cannot be expressed by the product of other polynomials different from itself or 1.

Some techniques for computing an inverse are:

- By checking every other element in the field (*bruteforcing*)
- Using the *Eulerian Extended Algorithm* (EEA)
- Computing $x^{(p-1)} \mod p$ or $x^{(F(x)-1)} \mod F(x)$
- Using the LS algorithm [56]
- Using the RS algorithm [56]
- Itoh-Tsujii algorithm [79] ($GF(2^m)$)

7.3.4 Exponentiation

Exponentiation i.e., computing $x^n \mod p$ given n is generally performed with a *square-and-add* algorithm, which consists in squarings and additons depending on the bits of the exponent. Improvements of such algorithm are the *Brauer algorithm* and the *Sliding window* approach.

7.4 Point Operations

Using the operations described in the previous section, it is possible to define the core operations on elliptic curves. These operations are used to realize the encryption, digital signature and all the other ECC protocols.

7.4.1 Point Addition

Point Addition is the core operation on EC. Given two points lying on the curve, say P and Q, the *sum* $R = P + Q$ is a point on the curve located as follows:

1. Given the EC formula, tracing the line intersecting both P and Q also intersects a third point R'.
2. The result R is obtained by R' and inverting its y coordinate.

An example is shown in Figure 7.3

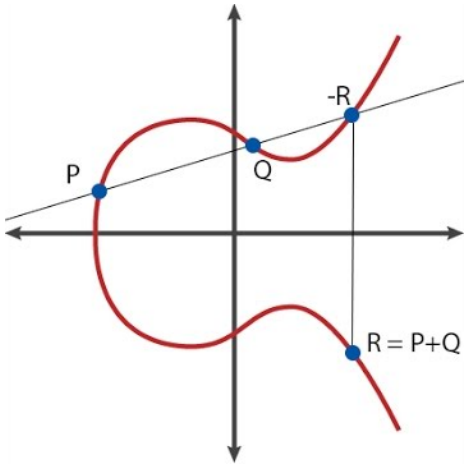


Figure 7.3: ECC Point Addition (in $\mathbb{R}^2$)

In Figure 7.4 the point addition as in $\mathbb{Z}/\mathbb{Z}_p$ is shown.

The basic formula involves the computation of the *slope* s. It can computed as:

$$s = \frac{\Delta y}{\Delta x} = (P_y - Q_y)(P_x - Q_x)^{-1}$$

Once s is computed, the resulting point R is equal to:

$$R_x = s^2 - x_P - x_Q$$

$$R_y = s(x_P - x_R) - y_P$$

7.4.2 Point Doubling

The case $P = Q$ is a special case in which the line through the P and Q points is indeed the *tangent* line in P. The computation of the slope adopted in the point addition in this case, cannot be applied. Instead, the derivative of the EC equation is computed to obtain $s = \frac{3x^2+b}{2y}$ for prime-curves and $s = x + \frac{y}{x}$ for binary curves.

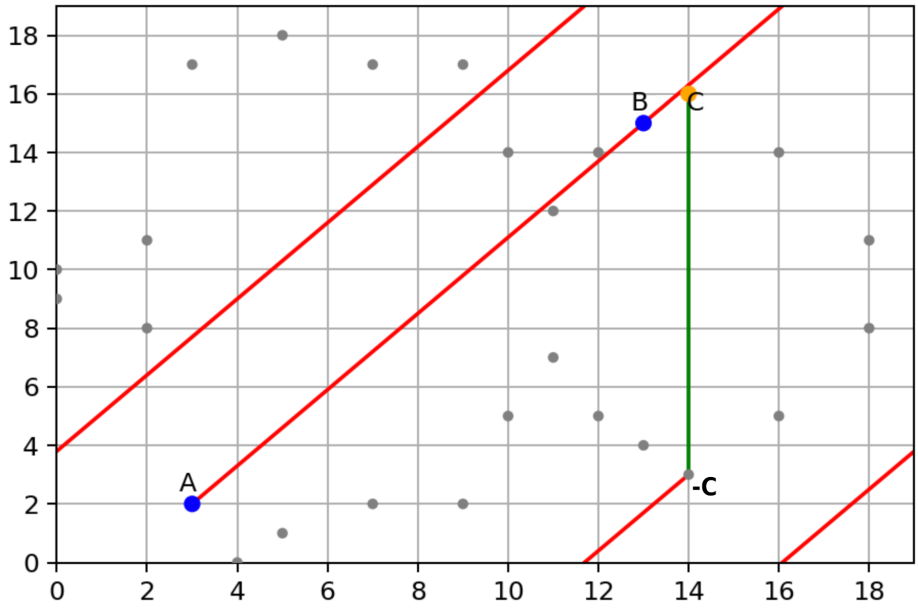


Figure 7.4: ECC Point Addition $C = A + B$(in $\mathbb{Z}/\mathbb{Z}_p$)

Then, the second part of the computation is equal to the point addition for prime-curves and, for binary-curves:

$$R_x = s^2 + s + a$$

$$R_y = s(x + R_x) + R_x y$$

7.4.3 Point Multiplication

Point Multiplication is the most important field operation in ECC. It is defined as the product of a scalar value k in the considered field and a point P lying on the curve:

$$R = kP = P + P + P + ... + P$$

The simplest way to compute such a product is to perform k additions. A better approach is to use the so-called *Double-and-Add* algorithm. An example is the following:

1. Initialize $R = \infty$
2. For each bit k_i in the binary representation of k, starting from the least-significant:
   (a) If the bit is 1, perform the point addition $R = R + P$. Otherwise, do nothing

(b) Perform the point doubling $R = 2R$

This algorithm works by splitting k in a sum of powers of 2 e.g.,

$$k = 456789 = 2^0 + 2^2 + 2^4 + 2^6 + 2^1 1 + 2^1 2 + 2^1 3 + 2^1 4 + 2^1 5 + 2^1 7 + 2^1 8$$

$$kR = 2^0 P + 2^2 P + 2^4 P + 2^6 P + 2^1 1P + 2^{12} P + 2^{13} P + 2^{14} + 2^{15} P + 2^{17} P + 2^{18} P$$

Since the performance of the point multiplication is critical, a vast number of optimizations are usually employed, including using pre-computed points, different multiplication schemes, alternative coordinate systems or different binary representations (e.g., the non-adjacent format, NAF [80]). Most of such optimization techniques will be briefly described in Section 7.7.

7.5 Elliptic Curve Discrete Logarithm Problem

Given the point multiplication $R = kP$. The *Discrete Logarithm Problem* (DLP) on elliptic curve (ECDPL) is the problem of finding k given R and P, which is demonstrated to be mathematically equivalent of computing a classical discrete logarithm on the field in which the curve is defined.

The naïve solution involves computing the point multiplication $R' = k'P$ for each value of $k' = 1...(n-1)$ until $R' = R$. However, if k is large enough, this *bruteforcing* approach is unfeasible due the time required.

All the ECC public-key protocols are based upon the ECDLP. The key pair is formed by a *private key* which is a scalar value in the numerical field chosen (e.g., $priv \in GF(p)$), while the associated public-key is just the point $P = privG$ where G is the point generator of the cyclic sub-group defined on the curve.

7.6 Protocols

The cryptographic protocols provided by ECC are built upon the operations described in the previous sections. The public-key encryption and decryption are provided by the *Elliptic Curve Cryptography Integrated Encryption Scheme* (ECIES), the digital signature forging and verifying are provided by the *Elliptic Curve Cryptography Digital Signature Algorithm* (ECDSA) and the (symmetric) key establishment protocol is provided by a Diffie-Helman adaptation over elliptic curves (ECDH). Also, protocols for the use of *Implicit Certificates* have been proposed, e.g., the *Elliptic Curve Mezenes-Qu-Vadstone* (ECMQV) protocol.

7.6.1 ECIES

In ECIES, a source (*Alice*) encrypts a *message* m with the public-key $Kpub_{bob}$ of the destination (*Bob*) so that the receiver, using its own private key $kpri_{bob}$, can decrypt the message. In addition, ECIES attach an *authentication tag* t that Bob can use to authenticate Alice's messages. The protocol is the following:

1. Alice generates a secure random number $0 < k < n$ and computes the point $R = kG$ which "stores" the k value while *hiding* it.
2. Alice computes also the point $Z = hkKpub_{bob}$, which has to be different from ∞. If not, Alice starts again from 1.
3. Using a *Key Derivation Function* (KDF) on R and Z, two symmetric keys key_1 and key_2 are generated.
4. Alice uses key_1 to encrypt m using a symmetric cipher, obtaining a ciphertext c
5. Alice uses key_2 to compute the authentication tag t (a *Message Authentication Code*)
6. Finally, she sends (c, R, t) to Bob
7. ...
8. Bob receives (c, R, t). First, he tries to recompute Z from $hkpri_{bob}R$. If $Z = \infty$, he discards the message
9. Using Z and R, Bob uses the KDF to recreate the keys key_1 and key_2
10. Bob checks the authentication tag with ey_2 and decrypt the message with key_1

7.6.2 ECDSA

ECDSA is used to create a *digital signature* s from a ciphertext (or from a plaintext, depending on the strategy adopted). The signature can be later verified to decide whether to accept or refuse a message. The signature is forged by using the sender private key $kpri_{alice}$ and thus can be verified by anyone (e.g., Bob) using Alice's public key $Kpub_{alice}$. The protocol is the following:

1. Alice generates a secure random number $0 < k < n$ and computes the point $R = kG = (r_x, r_y)$. If $r_x = 0$, Alice generate a new k
2. Alice uses an hash function H to obtain an hash of the message to sign: $e = H(m)$
3. The signature of the message is $s = k^{-1}(e + kpriv_{alice}r_x$
4. Alice sends Bob s and r_x
5. ...
6. Bob receives s and r_x and checks if $0 < r_x, s < n$. If the check fails, Bob refuses the message
7. Bob re-compute the hash of the message $e = H(m)$ and the following values:

- $w = s^{-1} \mod n$
- $u_1 = ew \mod n$
- $u_2 = r_x w \mod n$
- The point $X = u_1 G + u_2 Kpub_{alice}$. If $X = 0$, Bob refuses the message

8. Finally, Bob checks if $X_x \equiv r \mod n$. If so, the signature is verified and Bob can accept the message

7.6.3 ECDH

ECC also includes a key establishment protocol based on the Diffie-Helman approach. Once they have generated their key pairs, Alice and Bob exchange their public-keys. Then both multiply the public-key obtained with his/hers private key to obtain the same point:

$$P = kpriv_{alice} Kpub_{bob} = kpriv_{bob} Kpub_{alice}$$
$$= kpriv_{alice} kpriv_{bob} G$$

From this point, Alice and Bob take the x-coordinate P_x as shared key.

A variant of ECDH is the *ephemeral* version (ECDHE). The difference in ECDHE is that the key pairs used by the parties are temporary (i.e., generated on-the-fly) and not the *real* key pairs. This allows to avoid using directly the real keys but lose the ability to authenticate the parties via the transmitted public-keys.

An authenticated key establishment protocol alternative to ECDH/ECDHE is the *Elliptic Curve Mezenes-Qu-Vadstone* (ECMQV)[83].

7.6.4 ECQV

The *Elliptic Curve Qu-Vadstone* (ECQV) is a protocol for using *implicit certificates* to generate keys pairs. Let U the *identity* (expressed as a bit-string) of a target user and let the *Certificate Authority* (CA) key pair be (CA_{priv}, CA_{pub}). The CA, upon user request, issues the user's implicit certificate C_U which is a point on the curve. This certificate allows to extract the user public-key $Kpub_{user}$ once its identity U and the public-key of the CA that emitted the certificate are known.

The implicit certificate generation (and the user key pair) is performed as follows:

1. The user generates a secure random number $k \mod n$ and computes the point $R = kG$
2. The user sends R and its identity U to the CA
3. The CA chooses a secure random number k' and computes the implicit certificate $C_U = R + k'G$ and an *implicit signature* (used to verify the certificate) $\gamma_U = CA_{priv} + k * H(C_U || U)$. Those two values are sent back to the user

4. The user generate its new key-pair: its private key $kpriv_{user} = \gamma_U + kH(C_U||U)$ and its public key $Kpub_{user} = kpriv_{user}G$

The public key extraction can be performed as:

$$Kpub_U = CA_{pub} + H(C_U||U) \cdot C_U$$

which is correct, since:

$$CA_{pub} + H(C_U||U) \cdot C_U =$$

$$= CA_{priv}G + H(C_U||U) \cdot (R + k'G) =$$

$$= (CA_{priv} + (k + k')H(C_U||U))G =$$

$$= kpriv_{user}G = Kpub_{user}$$

Additional information on ECQV can be found in [84] and in [85].

7.7 ECC Optimizations

This section provides a non-exhaustive list of the common optimizations adopted in the ECC.

- **Barret Reduction**. This is an algorithm which can be used to avoid modular reduction by using the (approximating) equation $a \mod n = a - \lfloor as \rfloor n$ where $s = \frac{1}{n}$. The result accuracy depends on the accuracy of s.

- **Montgomery Reduction**. This reduction allows to move from a prime field to a (temporary) $\mathbb{Z}/\mathbb{Z}_{2^k}$. This allows modular reduction to be performed with fast bitwise AND operations.

- **Alternative Coordinate Systems**. Using different coordinate systems (e.g., projective coordinates) can reduce the number of complex operations (e.g., modular inversions), improving the performances.

- **Point Compression**. The elliptic curves are reflected along the x-axis, so, fixed a value for x, only two points (x, y) and $(x, -y)$ can be found. Using Point compression, points are represented by their x value and a single bit to indicate the *sign* of the y value (which can be re-computed by applying the elliptic curve equation). This drastically reduces (almost halving) the amount of memory required to store points.

- **Shamir's Trick**. When computing $aP + bQ$ (e.g., ECDSA), the *Shamir's Trick* can be used to perform the addition directly when applying the double-and-add multiplication algorithm on the multiplications, improving the performances.

- **Karatsuba Multiplication**. The *Karatsuba Multiplication* is a technique which can be used to reduce the number of word-sized multiplications. The idea is to split the binary representation of the scalars into two parts, e.g., $A = A_1||A_2$ and $B = B_1||B_2$, then the product $C = AB$ can be performed as $(A << n) + (Z << n/2) + B$ where n is the number of bit used to represent the numbers and $Z = (A_1 + A_2)(B_1 + B_2) - A - B$. This technique improves the performances when multiplications are slow while additions are fast.

- **Point Pre-computation (table)**. If the chosen elliptic curve has been fixed, it is possible to pre-compute sets of points in order to speed-up the multiplications. For example, given the base point G, by pre-computing $G, 2G, 3G, ...15G$ and storing them in a lookup table, it is possible to perform multiplications by scanning $n = 4$ bits at a time and using the lookup table to select the point to add to the accumulator.

- **Sliding Windows**. Similar to point pre-computation, but the points to pre-compute are those whose scalar has the most significant bit set.

- **Different Number Representations**. In order to reduce the total number of operations in point multiplication algorithms, alternative number representation can be used. A notable example is the *Non-Adjacent Form* (NAF) which is a signed bit representation. This representation allows to express numbers using less symbols, improving the performance and reducing the memory footprint when using also the pre-computation of points.

- **Alternative Curves**. Finally, it is worth mentioning that there exists other classes of elliptic curves which are designed to have better performances and other features. For example, the *Edwards Curves* ($x^2 + y^2 = c^2(1 + dx^2y^2)$), and the *twisted* version ($ax^2 + y^2 = 1 + dx^2y^2$). This latter class include the famous *Ed25519* curve [86].

Chapter 8

Intrusion Detection Systems for WSN

This chapter introduces the *Intrusion Detection System*s (IDSs), i.e., those systems, platforms, techniques and approaches aimed to detect intruders and/or incoming attacks targeting the network. Focus is put on the IDS targeting specifically resource-constrained platforms, in which common state-of-the-art approaches (e.g., *machine learning*-based [112]) are not feasible due the lack of computation resources and/or storage.

Overview on Intrusion Detection Systems — IDS are commonly classified by the kind of analysis (i.e., the detection model) they use to detect intruders. In particular, there exist two main families of IDSs: the *Anomaly-based* and the *Misuse-based* IDSs.

Anomaly-based IDSs try to detect behaviours which deviates from those expected for the target platform. In particular, an Anomaly-based IDS classify platform's behaviors into *expected* (normal) behaviors and *anomalous* behaviors. To do so, most of such IDSs use sets of rules and heuristics. Anomaly-based IDSs need also to be pre-*trained* to recognize the normal behaviors. This is also one of their weakness: the number of *false-positive* detections can represent an issue. Also, if the protected platforms has software bugs or vulnerabilities, attackers can still manage to carry out an attack while avoiding the platform to behave anomalously, hence avoiding the detection itself. A review on the common techniques adopted to classify anomalies can be found in Figure 8.1.

Misuse-based IDSs attempt to detect intruders by matching common attack *signatures* or *patterns*. Depending on the IDS, these signatures can be matched against the current state of the platform, the inputs, the outputs or on other sequences of events of the platform. Misuse-based IDS are usually fast, trading speed for the storage required to store all the needed signatures and patterns. Common examples of Misuse-based IDSs are the commonly used *Anti-virus* softwares used on unconstrained platforms. The main issue of Misuse-based IDSs is the detection of non-yet-known attacks, since no signature to be matched

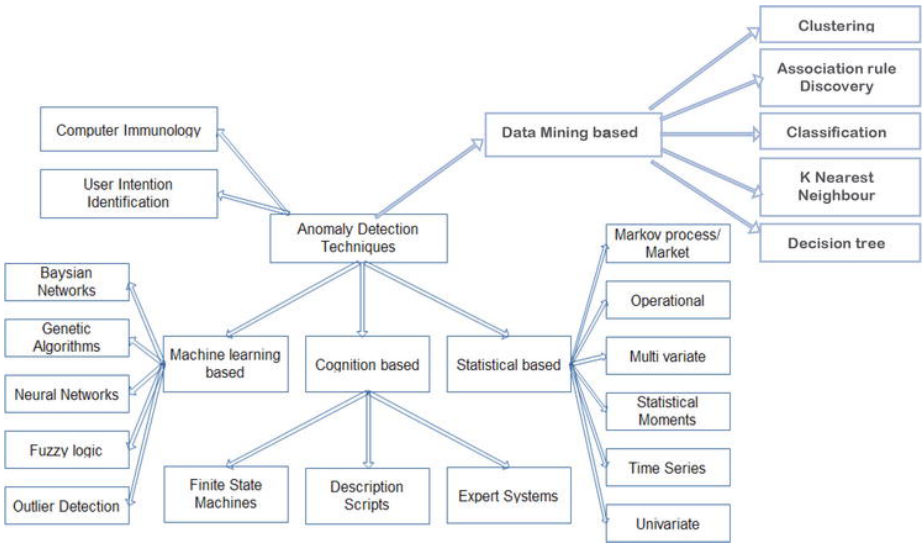


Figure 8.1: Common anomaly detection techniques [113]

is available. So, while the detection of well-known attacks is always successful, future attacks are hardly detected using signatures of other attacks. An example of misuse-based detection can be found in Figure 8.2.

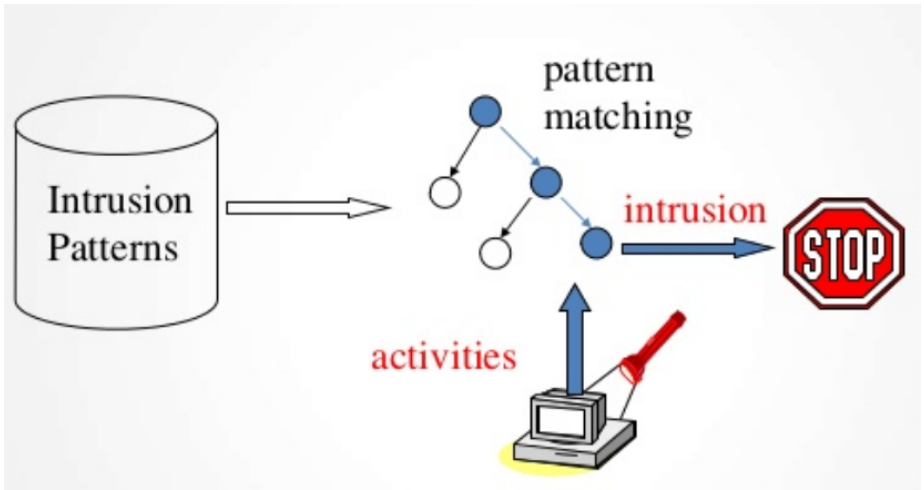


Figure 8.2: Example of misuse-based detection

Apart from the standard classification, there are *hybrid* IDSs which use a combination of the anomaly-based and misuse-base techniques or use non-standard solutions. *Specification-based* IDSs are a notable example of this kind

of IDSs. Also, in [104] the proposed IDS focuses on reducing the amount of consumed energy using a specific set of algorithms. Other hybrid approaches can be found in [105] [106] [107]. Other solutions, including a detailed comparative analysis of the techniques and IDSs for WSNs, can be found in [108].

Part II

Research activities

Chapter 9

The WSN security framework

The main objective of this work is to provide to WSN platforms an updated and enhanced *security framework* in respect of similar solutions e.g., [12]. The proposed framework is composed of a set of different sub-projects, each one addressing a particular security requirement or issue in WSNs.

Figure 9.1 shows the overall diagram of the framework.

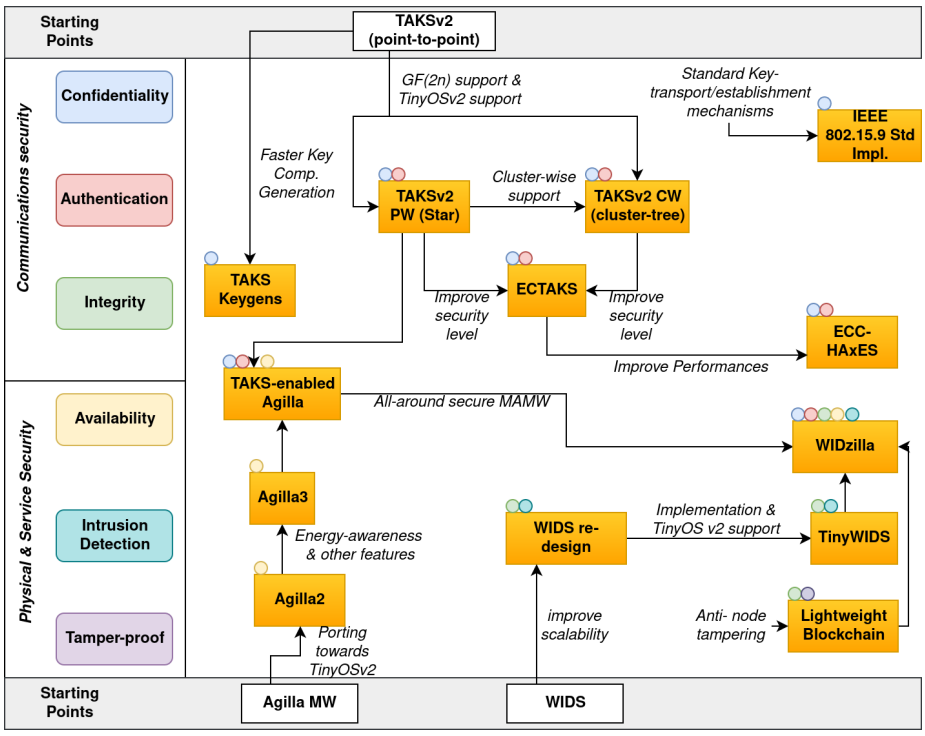


Figure 9.1: Overview of the proposed security framework

The security requisites addressed are:

- *Confidentiality*
- *Authentication*
- *Integrity*
- *Availability*
- (Active) *Intrusion Detection*
- *Data Tampering Protection*

In the proposed framework, *Confidentiality* and *Authentication* are addressed by using ad-hoc hybrid cryptographic schemes for WSN, which ensure good security and performances with a small memory storage impact. Such schemes have been enhanced to support different network (physical) topologies, new protocols, recent state-of-art cryptographic solutions and Hardware-based solutions. The research activities regarding these aspects are described in Chapters 10, 11 and 12

Despite *Integrity* features are already provided by the IEEE 802.15.4 standard, this thesis propose additional active *Integrity Checks* in the detection rules described in Chapter 13.

The framework, as already suggested by [18], uses a *MAMW* as software environment for WSN software applications. Using a MAMW allows network operators to manage running agents dynamically in both execution and movements. This ensures a high level of *Availability* since agents can be moved, replicated or re-injected on-the-fly. The research activities related to the evolution of the MAMW are described in Chapter 15.

In order to provide protection also for what concerns the *data* exchanged and stored, the framework includes a lightweight *blockchain-based* mechanism for providing the framework with an anti-tampering solution. As an additional feature, the proposed mechanism is able to detect and forbid communications to WSN nodes sending unreliable data. This anti-tampering mechanism is described in Chapter 14.

Intrusion detection in our proposed framework is provided by an IDS based on a enhanced version of [13]. The final IDS is merged with the MAMW and the previously described security solutions to an overall agent-based security solution. The research activities describing the evolution of the IDS and the design of the overall security platform are described in Chapter 13.

Chapter 10

TAKS

This chapter describes *TAKS*, a hybrid cryptography scheme designed for WSNs.

10.1 Motivation

In resource-constrained devices as the WSN nodes, providing an efficient cryptographic scheme is crucial, since it represents the first and most important defence to protect the WSN *confidentiality*. Symmetric schemes, thanks to their performance and low resource requirements, are the perfect candidates for protecting and concealing data and messages from malicious third parties. However, as mentioned in Section 6.2, the lack of a secure mechanism for exchanging the symmetric key makes symmetric schemes alone not enough to protect the WSN node. This is the so-called *Key Distribution* problem.

In literature, the *Key Distribution* problem is addressed by various works. The *default* solution is to adopt the *Key Agreement* or Key Establishment protocols (e.g., [47]) which are based on public-key cryptographic schemes (e.g., [116][117]). However, given the resource-constrained WSNs nodes, this solution is generally avoided, since it could cause unbearable performance losses, unsatisfactory throughput, increased energy consumption or an excessive impact on memory.

Another common solution to the *Key Distribution* problem to *pre-distribute* the keys to WSN nodes during their installation. This solution is efficient but, at the same time, makes the WSN nodes and the symmetric scheme vulnerable, since if an attacker manages to capture a node and read the symmetric key from it, the security is defeated.

Other solutions proposed in literature address the problem by adopting ad-hoc techniques that take advantage of the WSN features. For example, in [121] the authors propose a pre-distribution key management scheme which aims to establish *group*s key inside the WSN. This is useful for *cluster-tree* WSNs since the so-called groups could be mapped to the WSN clusters.

Among the ad-hoc techniques, the *hybrid cryptography* is one of the most

promising, since it combines the benefits of both the symmetric cryptography (performances, small memory footprint, etc.) and public-key cryptography (key management, authentication, etc.) into a single scheme. Examples of hybrid cryptographic schemes for WSNs are [19][119][119]. Moreover, a systematic review on hybrid cryptography schemes is proposed by authors in [118].

Given the advantages of hybrid cryptography, it has been selected to solve the *key exchange* problem and to provide *confidentiality* to WSN nodes. In particular, given the previous works its compatibility with TinyOS and the possibility to directly *authenticate* messages, *TAKS* [19] has been chosen.

The next sections briefly resume TAKS and the thesis' research activities aimed to create an improved and enhanced version of it.

10.2 TAKS Introduction

The *Topology-Authenticated Key Scheme* (TAKS) [19][20] is a hybrid cryptography scheme which exploits *vector algebra* to provide a lightweight mechanism for distributing keys, encrypting and authenticating messages in WSNs.

The idea behind TAKS is to use partial key components (called *Key Components*) and distribute them according to a *logical topology*. The components can be combined together to re-construct the full cryptographic key for encryption and authentication. Only by combining the right set of components (i.e., the components owned by the nodes which are allowed to communicate in the defined logical topology) the right key is generated by both the sender and the receiver of a message. This allows to:

- reduce the amount of memory used to store keys, since in TAKS only the components are stored;
- defend against attackers, which cannot guess the key without having already compromised both the parties involved in a communication.

The current version of TAKS is *2* (TAKSv2) [20]. The main difference with the first version ([19]) is the number of steps required to complete a communication (in TAKSv1, two steps are required, while in TAKSv2 all is accomplished with one step).

In this thesis research activities, starting from [19], [10], [20] and [11], TAKS has been extended, implemented in different versions and evaluated its security and performances in different scenarios. All of those are described in the following sections.

10.3 Definitions

In TAKS, the cryptographic keys are constructed starting from *key components*. There exists the following key component types:

- The *Local Key Component* (**LKC**) is a private component which is kept secret by its owner.

- The *Transmit Key Component* (**TKC**) is a public component. The TKC is known by all the parties which are enabled to exchange messages with its owner (in the chosen logical topology).

- the *Topology Vector* (**TV**) is a public component. It coincides with the TKC in simpler configurations (e.g., pair-wise schemes).

Each component is a vector of d components. For example, if $d = 3$, $LKC = (l_1, l_2, l_3)$ where each l_x is a value in the mathematical field considered (e.g., $GF(2^k)$ with an *irriducible* polynomial $p(x)$) The set of all the components stored in a motes (along some additional metadata) constitutes the *Local Configuration Data* (LCD).

The combination of the key components is performed by the so-called TAK function. This function is not fixed: every function defined according the prerequisites described in [20] can be a valid TAK function. The default TAK function adopted is the vector *cross-product*. The result of the TAK function is a *Shared Secret* (SS), which in TAKS is used to derive the keys used for encryption and authentication. Symmetric encryption is performed via the so-called ENC function. TAKS can use any stream or block cipher as ENC function. Section 10.4 describes the research activities on providing TAKS implementations also a IEEE 802.15.4-compliant encryption function. The authentication function, called $AUTH$, is a verification function which can be based on any cryptographic *Message Authentication* function (MAC). Following the IEEE 802.15.4 standard, the $AUTH$ function is defined to be the comparison of the AES CBC-MAC computations.

The message format (i.e., the IEEE 802.15.4 MAC payload) to be used to support TAKS consists in three fields:

- the ciphertext (c) obtained by encrypting the plaintext with the chosen symmetric cipher with the SS as key;

- the authentication tag τ, obtained by the MAC function using SS as key;

- the *Key Reconstruction Information* KRI, which is the TKC of the sender obfuscated by the *nonce* value α.

Finally, depending on the *physical* topology of the WSN and the intended communication directions, TAKS can be used in *Pair-Wise* mode (PW) [19] or in *Cluster-Wise* mode (CW).

10.3.1 Pair-Wise scheme

In a PW scheme, only the LKC and TKC components are used (since $TKC = TV$). A private LKC is assigned to each mote in the WSN along the TKCs of the motes enabled to communicate with the considered mote in the defined topology.

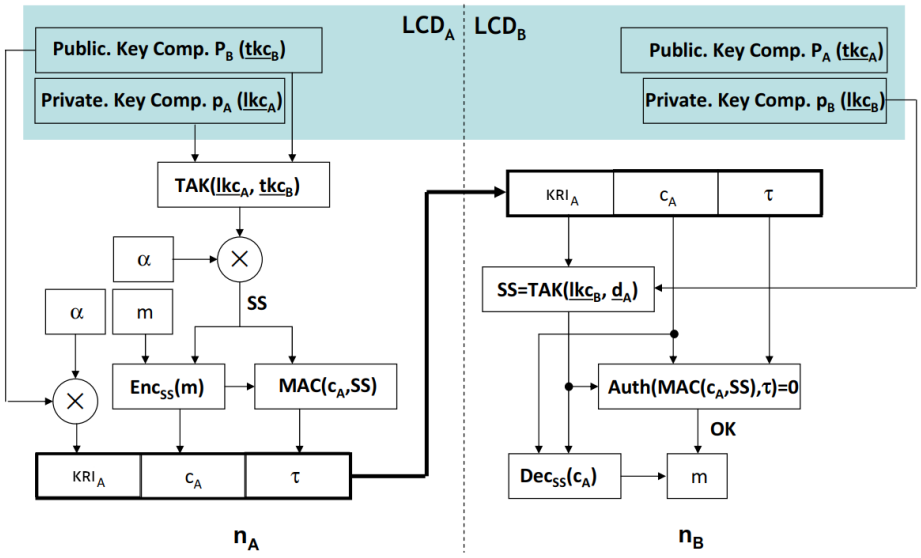


Figure 10.1: TAKSv2 Pair-Wise scheme

The PW scheme is shown in Figure 10.1. The TAK function computed by mote i for sending a message to mote j is the following (considering $d = 3$):

$$TAK_{i \to j} = \alpha * LKC_i \times TKC_j = \alpha TKC_i \times LKC_j = KRI \times LKC_j$$

$$= \alpha * \begin{Vmatrix} \hat{i} & \hat{j} & \hat{k} \\ lkc_1 & lkc_2 & lkc_3 \\ tkc_1 & tkc_2 & tkc_3 \end{Vmatrix} = \begin{Vmatrix} \hat{i} & \hat{j} & \hat{k} \\ lkc_1 & lkc_2 & lkc_3 \\ kri_1 & kri_2 & kri_3 \end{Vmatrix}$$

10.3.2 Cluster-Wise scheme

The TAKS-CW scheme is a variation designed to work on *cluster-tree* physical topologies. In this scheme, the TAK function results for communications from the cluster-head to the cluster members has to be independent from the destination member [20]. To do so, the key components are combined as follows:

$$TAK_{i \to j} = \alpha * LKC_i \times TV_{i \to j} = \alpha TKC_i \times LKC_j = KRI \times LKC_j$$

$$= \alpha * \begin{Vmatrix} \hat{i} & \hat{j} & \hat{k} \\ lkc_1 & lkc_2 & lkc_3 \\ tv_1 & tv_2 & tv_3 \end{Vmatrix} = \begin{Vmatrix} \hat{i} & \hat{j} & \hat{k} \\ kri_1 & kri_2 & kri_3 \\ lkc_1 & lkc_2 & lkc_3 \end{Vmatrix}$$

10.4 TAKS Enhancements

10.4.1 Flexibility in TAKS key component sizes

In the previous works [19][10][11][20], the proposed TAKS versions uses tridimensional vectors as key components (i.e., $d = 3$). Although the previous works describe the vector sizes to be variable, no additional information are provided. During the thesis research activities, other possibilities for vector dimensions have been investigated. In particular:

- $d > 3$ i.e., iper-vectors. In this configuration, the key components are longer but the security level is enhanced. The regular TAK computation has no issue and it can be used to compute the SS normally
- $d = 2$ i.e., bidimensional vectors. In this configuration, the key components are shorter but the cross-product cannot be computed as usual (i.e., there not exists two-dimensional vector which is perpendicular to two two-dimensional vectors lying on a plane)

In order to solve the cross-product issue in $d = 2$ case, the chosen approach has been to extend the two key components by adding a third component which is assumed to be 0. In this way, when computing the TAK function, the result will be a tridimensional vector with only one vector component different from 0. This third component will be used as SS.

10.4.2 Random number generation

In the previous works, no additional information is provided on a very important aspect: the generation of the α nonce value. On standard computing platforms, a *Cryptographically-Secure Pseudo-Random Number Generator* (CSPRNG) is fed with a *seed* value coming from a source of *entropy* to produce non-replicable random numbers (i.e., an attacker is unable to re-create the random number sequence obtained by the CSPRNG). Since in WSN nodes there is no good source of entropy for generating seeds to be used to generating random numbers, there is the need for ad-hoc approaches.

This thesis provides the following solutions:

- A modified version of the famous *Mercenne Twister* [32]. This is a lightweight version which is optimized for WSN nodes both in performances and storage footprint.
- A lightweight implementation of the *Blum-Blum-Shub* CSPRNG [33].
- A *seed* generator, which combines:
  - the current values of the free-running timers on the WSN node
  - the current battery voltage (expressed in ADC steps)
  - the current values on the available sensors (expressed as ADC steps for analog sensors)

– the current EM energy measurements coming from the radio tranceiver

Although the proposed solution do not solve entirely the issue, it increases the resilience of TAKS in respect to nonce-based attacks, timing-synchronization attacks, etc.

10.4.3 Symmetric Encryption

In the previous works, the *symmetric cipher* adopted in TAKS (AES-128 bits in CCM* mode) is not detailed in its implementation. In order to clarify and enhance this component, this thesis propose a new optimized AES-128-CCM* implementation in software, then, for the WSN nodes platforms using the *CC2420* radio chip, the approach proposed in [54] has been followed to provide a software wrapper around the hardware implementation available in the transceiver. In this way, it was possible to obtain better performances and reduced the memory storage (e.g., AES tables). Note that, in this version, since the encryption and authentication are performed directly on incoming frames (which are hidden from the MCU if decryption/authentication fails) TAKS needs to split each communication in two steps, one for exchanging KRI to allow both parties to re-construct and set the key inside the CC2420 transceiver and one for the actual communication.

10.4.4 TAKS Key Generation

One aspect not yet described in this chapter is *how the key components are actually generated*. In the previous works, key components are computed with specialized Matlab-based scripts once the logical topology is known and inserted as input one node/link at a time.

In order to automatize the key component generation, a new approach has been provided. Given the number of nodes and the key length, all the possible components combinations are created (both for the pair-wise and the cluster-wise scheme). Developers can, then, take only the components needed in the chosen logical topology.

The idea behind the new generator is the following. In the pair-wise version of TAKS, it has to be that:

$$\alpha LKC_i \times TKC_j = SS = \alpha TKC_i \times LKC_j$$

To achieve this result, a random LKC is generated for every node in the WSN; then a random target SS for each possible pair is computed and by *inverting* the TAK function and the TKC of the both nodes in the pair is obtained.

A similar approach is followed for the cluster-wise version. Given that:

$$\alpha LKC_i \times TV_{i \to j} = SS = \alpha TKC_{i \to j} \times LKC_j$$

we generate the LKC for every node, the SS and TV for every possible cluster/pair and finally compute the proper TKC to verify the equation.

The Python code for the generator is available at the author's GitHub page [55].

10.5 TAKS Implementations

This section describes all the implementations of TAKS in previous works and the new enhanced version resulted from the presented research activities.

10.5.1 TinyOS 1.x implementation

The first TAKS implementation (based on TinyOS 1.x and the NesC language) is described in [11] and in [20]. Here, the pair-wise version of TAKS (using AES 128bit CBC + CBCMAC) is described and pseudo-code is provided. This implementation is based on the `GenericComm` component of TinyOS 1.x. No proper MAC layer is used.

This version was developed for MicaZ motes and ceased to work after the introduction of TinyOS 2.x.

10.5.2 TinyOS 2.x TKN154-enabled and Atmel-based implementations

This second version is a major review of the previous. It is a re-written version compatible with TinyOS 2.x, supporing a larger number of platforms and using the TKN154 MAC layer and the Atmel 802.15.4 proprietary MAC layer. The overall architecture has been improved and made more flexible for later enhancements (e.g., adopting specific software engineering design patterns). This implementation of TAKS supports the pair-wise scheme and the star-topologies. Additional information can be found in [3].

10.5.3 Cluster- and Mesh-enabled implementation

Starting from the previously described implementation as baseline, a third implementation of TAKS has been provided in the context of the SEAMLESS project. This version, in particular, enables support for a TKN154-based cluster-tree topology and generic mesh topologies along with a routing algorithm. Additional information on SEAMLESS are reported in Chapter 17.

10.5.4 New implementation

Starting from the previous work and considering all the enhancements described in Section 10.4, a new versions of TAKS has been released. This new version is compatible with all the major WSN node platforms, has a very limited memory footprint and better performances. Also, CC2420's hardware based encryption is supported for the WSN node equipped with it.

This new version has been evaluated and Figure 10.2 shows the performance results while Figure 10.3 shows the memory footprint obtained when adopting TAKS on a sample TinyOS application.

WSN Mote	MCU	Keysize [bits]	Encryption	Decryption
Advanticsys CM5000 (compiler)	MSP430F1611 *msp430-elf-gcc v7.3*	128	47.6 ms	8.4 ms
		192	74.2 ms	14.8 ms
		256	92.4 ms	15.6 ms
Advanticsys XM1000 (compiler)	MSP430F2816 *msp430-elf-gcc v7.3*	128	32.2 ms	5.8 ms
		192	49.8 ms	10.6 ms
		256	63.2 ms	11.6 ms
MEMSIC IRIS (compiler)	ATMega1281 *avr-gcc v 8.2*	128	3.1 ms	0.56 ms
		192	5.56 ms	1.84 ms
		256	5.8 ms	1.04 ms
MEMSIC MicaZ (compiler)	ATMega128L *avr-gcc v 8.2*	128	3.09 ms	0.56 ms
		192	5.56 ms	1.81 ns
		256	5.88 ms	1.04 ms

Figure 10.2: New TAKS version: performances

Memory Overhead	ROM (bytes)	% ROM	RAM (bytes)	% RAM
MicaZ	+4490	3,43%	+590	7,20%
Iris	+4582	3,50%	+800	9,77%
telosb	+13618	27,71%	+549	6,70%
xm1000	+13660	27,79%	+549	6,70%

Figure 10.3: New TAKS version: memory footprint

10.6 TAKS IEEE 802.15.9 KMP

In order to take advantage of the introduction of the IEEE 802.15.9 standard to solve the key transport issues of the IEEE 802.15.4, the research activities focused on creating also an adapted version of TAKS which adhere the interfaces proposed by the IEEE 802.15.9 standard. In particular, given the structure of the layer introduced by the standard (see Figure 5.2) TAKS was adapted to be a valid *KMP* component. The adaptation grants IEEE 802.15.9-enabled WSN the possibility to adopt the TAKS topology.authenticated keys, overcoming the necessity of a strict key management scheme.

In order to adapt TAKS, the following mapping between TAKS mechanisms and the KMP interfaces has been created:

- The *KMP-CREATE* interface is used to establish a secure connection (or, in IEEE 802.15.4 jargon, a *Security Association* (SA)). TAKS, in order

to provide a compatible functionality, uses the primitives of this interface (*request, confirm, response* and *indication*) to exchange the *Shared Secret*. The idea is that the sender, using the *KMP-CREATE.request* primitive, generates the KRI to be sent to the receiver. The receiver, once a *KMP-CREATE.indication* is received, uses the KRI re-construct the *Shared Secret*.

- The *KMP-FINISHED* is used to inform the requester (*KMP-CREATE.request*) that the SA is established. TAKS, uses the *KMP-FINISHED.indication* to inform both sender and receivers that the *Shared Secret* has been successfully reconstructed.

- The *KMP-DELETE* interface is used to remove a previously established SA. TAKS does not use such interface, but it could be used in more complex scenarios as consequence of the *de-association* of a WSN node from the WSN.

- The *KMP-PURGE* is used to abort a current key creation/deletion. In TAKS it is not implemented.

As for the *Information Elements* involved, TAKS uses a reserved KMP ID of 255, since it is not yet proposed for registration to the IEEE registrar authorities.

In order to complete this research branch, a future work is to provide also a lightweight instantiation of the IEEE 802.15.9 MPX layer. In this way, a full (and *world-first* at the time of writing) IEEE 802.15.9-compliant stack implementation could be provided to the research community.

10.7 TAKS-enabled Open-ZB

The *OpenZB* project [67] is the combination of a ZigBee protocol implementation with a IEEE 802.15.4 MAC implementation for TinyOS-based applications. In this version of TAKS, apart from supporting the AES-128-CCM via the CC2420 transceiver (see [54]) the security aspects are not fully implemented (e.g., key management). In order to provide a lightweight cryptographic scheme supporting key management, this thesis proposes a prototype implementation of TAKS on top of OpenZB.

This implementation takes advantages of the fact that OpenZB uses a MAC layer which is very similar to the TKN154[1]. So, the TK154-based TAKS version has been used and adapted to the OpenZB codebase so that every data frame (`MCPS-DATA.indication`) is encrypted and decrypted according using TAKS. Finally, we provided also a variant in which the TAKS encryption/decryption happen at the NWK layer, targeting the `NCPS-DATA.indication` data "packets" instead of the MAC frames.

[1] In fact, OpenZB contributed to the current TKN154 implementation

10.8 Related publications

Details on the TAKS enhancements obtained from authors' research activities can be found in [3] and in [5].

Chapter 11

ECTAKS

This chapter describes the evolution of TAKS in the direction of elliptic curve cryptography: *ECTAKS*. Here the idea behind ECTAKS is introduced along with the supported protocols and the result of the research activity: a working implementation based on TinyOS 2.x and the TinyECC project [35].

11.1 Overview

In [14] ECTAKS has been proposed as an evolution of TAKS towards ECC. ECTAKS grants a better security level and, thanks to the ECC, reduced key sizes. The main idea behind ECTAKS is to map TAKS mechanisms on ECC standard protocols so that the TAK function result is used as scalar for creating an ECC key pair.

11.2 Vector Operations

In ECTAKS [14], new vector product operations are introduced to compute the final *Shared Secret*. The first one is the *scalar vector by point* product: it computes a *vector of EC Points* by multiplying each vector component by the point operand:

$$(s_1, s_2, s_3, ..., s_n) \cdot P = (s_1 P, s_2 P, ..., s_n P)$$

The second operation, the *scalar vector by point vector* product, computes a vector of EC points performing a component-by-component multiplication:

$$(s_1, s_2, s_3, ..., s_n) \times (P_1, P_2, ..., P_n) = (s_1 P_1, s_2 P_2, ..., s_n P_n)$$

In the next sections the variation introduced in the standard ECC-based encryption and digital signature protocols are analyzed.

11.3 Research contribution

In [14], only the construction of the key components is thoroughly described. The protocols constructed on top of ECC and adapted to follow TAKS mechanisms has not been developed.

Pugliese et al. developed two prototype of the ECTAKS' *Integrated Encryption Scheme* (ECTAKS-ECIES) and the ECTAKS' *Digital Signature Scheme* (ECTAKS-ECDSA). In the thesis research activities focused on refining these protocols and implementing them using the *TinyECC* library as baseline.

11.3.1 ECTAKS-ECIES

In order to adapt the TAKS hybrid mechanism to ECC, the key components LKC and TKC are initially mapped to the private key p_{ij} and public key P_{ij}. ECTAKS enforces that:

$$(\alpha LKC_i \times TKC_j) = tak_{ij} = p_{ij}$$

and:

$$tak_{ij}G = p_{ij}G = P_{ij} = ECTAK_{ij}$$

In order to match the ECC bi-dimensional representation, the dimension of the TAKS vectors is $d = 2$ and the *adapted* cross-product operation is used (see Section 10.4).

In ECIES, data is encrypted with the public-key and decrypted with the private key. The encryption is performed with a symmetric cipher and a key obtained by a *Key Derivation Function* computed on $R = \alpha G$ and the point Z which, during encryption is computed as $Z = \alpha h P_{ij}$, while in decryption is computed as $Z = hpijR$. The result of the protocol is a triple (c, R, τ) where c is the ciphertext, τ is the authentication tag and R is a point used to reconstruct the symmetric key.

In ECTAKS, this mechanism is modified as follows. R is constructed to be the equivalent of the TAKS' KRI in following way:

$$R = \alpha TKC_j \cdot G$$

while Z, in the encryption case:

$$Z = \alpha h ECTAK_{ij} = \alpha h TAK_{ij} G = \alpha h (LKC_i \times TKC_j) G$$

and in the decryption case:

$$Z = hLKC_j \cdot R_{received}$$

In both cases, the resulting SS used for encryption/authentication tag is obtained by $SS = KDF(Z, R)$. The overall ECTAKS-ECIES protocol is the following.

Encryption:

1. select a random integer α
2. compute $ECTAK_{ij} = (LKC_i \times TKC_j)G$
3. compute $R = \alpha TKC_i \cdot G$
4. compute $Z = hECTAK_{ij}$. If $Z = 0$, go back to 1
5. compute $SS = (SS_1, SS_2) = KDF(Z, R)$
6. $c = Encryption(m, SS_1)$
7. $\tau = MessageAuthentic(c, SS_2)$
8. return (c, R, τ)

Decryption:

1. compute $Z = hLKC_j \cdot R = \alpha ECTAK_{ij}$. If $Z = 0$, refuse the message
2. compute $SS = (SS_1, SS_2) = KDF(Z, R)$
3. $\tau' = MessageAuthentic(c, SS_2)$
4. check if $\tau = \tau'$. If different, refuse the message
5. $m = Decryption(c, SS_1)$
6. return m

11.3.2 ECTAKS-ECDSA

The variation on the ECDSA protocol is a bit more complex. Given the secure hash of the message e, the signature of the message m from node i is created as follows:

1. select a random integer α
2. compute $R = kG$ and $r = R_x$. If $r = 0$, go back step 1
3. compute S so that $S_x = k^{-1}(e + LKC_{i_x} r)$ and $S_y = k^{-1}(e + LKC_{i_y} r)$
4. return (r, S).

The signature can be verified by node j as follows:

1. check if $1 \leq r \leq n - 1$ and if $1 \leq S_* \leq n - 1$. If not, refuse
2. compute $z = S \cdot TKC_j = (k^{-1}(e + TAK_{ij_x} r), k^{-1}(e + TAK_{ij_y} r))$
3. compute $w = z^{-1}$
4. compute $u_1 = ew(TKC_{j_x} + TKC_{j_y})G$
5. compute $u_2 = rwTAK_{ij}$
6. compute $X = u_1 G + u_2 G$
7. check if $X_x = r$. If so, accept the signature, otherwise refuse it.

11.4 Implementation

As stated before, the thesis research activities on ECTAKS started by collecting information on the state-of-the-art of ECC software libraries (e.g., *OpenSSL* [109]). however, do not fit the requirements imposed by WSN platforms. The only library which address the WSN constraints while offering a large set of optimizations is *TinyECC* [35]. Although TinyECC is not the best available in terms of performances and memory footprint, it is still one of the few libraries which is almost independent on the (prime) curve selected. Other libraries, instead, offer better performances in exchange of a limited set of available curves (often, just one).

Since TinyECC did not support all of available WSN node platforms, the first part of the proposed implementation focused on adding such support (in particular, for the *Iris* WSN platform). The resulting version of TinyECC supporting the Iris motes (compared with the similar *MicaZ* platform) is shown in Figure 11.1. A second pre-implementation step has been to update the set of

Operation (time in ms)	Time for MicaZ As in [35]	Time measured On IRIS	%
TinyECC init	N/A	2673	N/A
PublicKey Generation	2961,8	2761	93,22%
ECIES init	2639,95	2727	103,30%
Encrypt	6549,62	5991	91,47%
Decrypt	4198,64	3865	92,05%
ECDSA init	5094,01	5203	102,14%
Sign	3175,17	2944	92,72%
Verify	4045,41	3718	91,91%
ECDH init	2635,65	2625	99,60%
Establish	3509,83	3230	92,03%

Figure 11.1: TinyECC support for Iris platform: results in comparison with the MicaZ

available elliptic curves to contain *only* the curves which are currently considered secure according to [77].

Then, the implementation has proceeded as follows:

1. The two new product operations have been implemented in a new NesC component

2. Two new components have been added, one for ECTAKS-ECIES and the other for ECTAKS-ECDSA

3. Following the description provided in the previous sections, the variations introduced in ECTAKS have been implemented

4. The test components of TinyECC have been updated to support the ECTAKS components

5. Finally, the whole TinyECC code has been cleaned up and re-organized to put the focus on protocols.

11.5 ECMQV

The *Elliptic Curve Mezenes-Qu-Vadstone* [127] (ECMQV) is an authenticated key exchange protocol largely adopted as alternative to Diffie-Helman based approaches thanks to its resistance to active attackers. It is also adopted in the some ZigBee profile specifications [128] as main key exchange protocol. A brief overview on how this protocol works follows. Alice and Bob start with their own key pairs (A, al) and (B, bo). They generate (each one) a new key pair, i.e., (K_A, k_a) for Alice and (K_B, k_b) for Bob. Then, Alice computes $S_a = k_a + \overline{R}(K_A)al$ while Bob does the same with $S_b = k_b + \overline{R}(K_B)bo$, using $\overline{R}$ as the function that, given a point on the chosen elliptic curve, returns the L least significant bits of the x coordinate of the point. At this point, Alice and Bob send each other K_B and K_A. Finally, they can compute $K = h*S_a(K_B+\overline{R}(K_B)*B)$ and $K = h * S_b(K_A + \overline{R}(K_A) * A)$ to retrieve the (shared) secret K, since:

$$
\begin{aligned}
K &= hS_a(K_B + \overline{R}(K_B) * B) \\
&= h * S_a(k_b * G + \overline{R}(K_B) * bo * G) \\
&= h * S_a(k_b + \overline{R}(K_B) * bo) * G \\
&= h * S_a * S_b * G \\
&= h * (k_a + \overline{R}(K_A) * al) * S_b * G \\
&= h * (k_a * G + \overline{R}(K_A) * al * G) * S_b \\
&= h * (K_A + \overline{R}(K_A) * A) * S_b = K
\end{aligned}
$$

In comparison with ECMQV, using the topology-authenticated keys, once the key components have been deployed, ECTAKS has no need for a key exchange phase to obtain a suitable symmetric key at the price of less resistance against active attackers.

11.6 Results and Future Works

Current ECTAKS source code is available at author's GitHub page [63]. A series of tests has been conducted to validate ECTAKS and evaluate its performance. However, the first results demonstrated that, at the current state, ECTAKS has unsatisfactory performance: for example, using a 192 bit curve, a single point multiplication (e.g., $TAK\ G$) took over 30 seconds. Although this performance issue is still under investigation, other solutions are currently under examination, including the shift from a software implementation to a *hardware-accelerated* implementation, as it is discussed in the next Chapter.

Apart from the performance aspects, another planned improvement is the support for different type of EC. In particular, considering the updated list of secure curves in [78], a future work will be adding the *safe* curves in the set of supported curves.

Chapter 12

ECC-HAxES

This chapter describes the *ECC Hardware Accelerators for Embedded Systems* (**ECC-HAxES**). A basic introduction on the motivations and the design considerations is reported in Section 12.1, while in the following sections the design and its implementation are discussed. Finally, the validation step and the performance analysis are described.

12.1 Overview

The implementation of public-key cryptographic protocols on the WSN mote is often unfeasible due the hard performance and storage limitations. In order to overcome this issue, one solution is to use an application-specific *hardware accelerator* suitable for an energy-constrained device. In this context, a hardware accelerator is meant to provide a boost in terms of performance by moving a computation or part of it from software to hardware via e.g., FPGA platforms or ASIC designs.

There exists various hardware designs for cryptography-oriented hardware accelerator, in particular for ECC. In order to be as light and performant as possible, most of them adopt *compromises* which, in most of the cases, restrict the flexibility (e.g., by fixing the used curve) of the accelerator.

For example, in [57] a full crypto-processor is designed to provide ECC over $GF(2^{163})$. The solution has been verified and evaluated on multiple FPGA platforms, however, the flexibility of the hardware is very limited, since the inner circuits have been designed only to work with the standard 163-bit curves (*sect163k1* and *sect163r1*).

Shahid et al. [58] present a ECC co-processor featuring a execution unit, composed by an *Arithmetical and Logic Unit* (ALU) with multiple modular multiplier, and a *scheduler*. Despite this solution grants a good level of flexibility, its area occupation grows steadily and it is not in line with our *pre-requisites* (see below).

The area issues are addressed by solutions as [59], where authors describe

a ECC accelerator supporting *dual field* arithmetics i.e., both 160-bit prime curves in $GF(p)$ and binary curves in $GF(2^{163}$. This platforms uses a set of instructions, a control unit and an ALU composed by multiple clusters. The accelerator was implemented in 90-nm CMOS technology with interesting results in the final area occupation.

In respect of the state-of-art solutions, this thesis proposes *ECC-HAxES*, an ECC hardware accelerator designed with the following requirements and considerations in mind:

1. All-around *flexibility*: it shall be possible to switch to a different elliptic curve at runtime, different mathematical field or different features with the least possible number of fixed parameters. Flexibility, however, come with a price: most of the state-of-art solutions fix the curve so that it is possible to easily perform off-line pre-computations, which translates to improved and optimized algorithms. The challenge is to obtain both flexibility and high performances.

2. The design is also focused on a *minimum area occupation*. While this requirement could result also in non-optimal performances[1], it allows to greatly reduce costs and open the way to design small break-out board (and small and low-power FPGA chip) which dimensions are comparable to the embedded system using the accelerator. In this way, for example, a WSN mote would be still small and cost-effective.

3. It should provide *on-demand* services. The accelerator should be, when unused, in a reduced-consumption state (e.g., stand-by) but ready to switch to an *active* mode when required. In this way, even with the adoption of this hardware accelerator, energy consumption could be limited. Ideally, since the accelerator operations are quicker than their software counterpart, the combination of an embedded system and a hardware accelerator could event be more energy-efficient than the pure software solution when computation-intensive tasks are considered (e.g., ECC).

4. The accelerator should provide the best possible *performances* along with a good *design quality*, so that it would be easy to perform post-release modification to fit different requirements.

5. To reduce costs and development time, the hardware accelerator is (at least, initially) developed for reconfigurable hardware platforms (e.g., FPGAs) using a hardware design language (HDL) when implementing the design.

6. Since the hardware accelerator needs to be used by a separated embedded system, a custom *communication protocol* over a common serial interface (e.g., UART, SPI, I2C) shall be defined. The protocol shall define the

[1] those performances, in every case, would be orders of magnitude better than the software counterpart

format of the messages used to exchange commands and data between the embedded system and the hardware accelerator.

The following sections describe our ECC hardware accelerator, its design, its implementation and the results in terms of area and performances.

12.2 ECC-HAxES

ECC-HAxES basic idea is to provide the implementation of ECC public-key encryption/decryption and digital signature creation/verification to a (client) embedded system. In particular, ECC-HAxES aims to provide the implementation of the *Elliptic Curve Integrated Encryption Scheme* (ECIES) and the *Elliptic Curve Digital Signature Scheme* (ECDSA) as basic services.

Figure 12.1 shows an example (*digital signature request*) scenario.

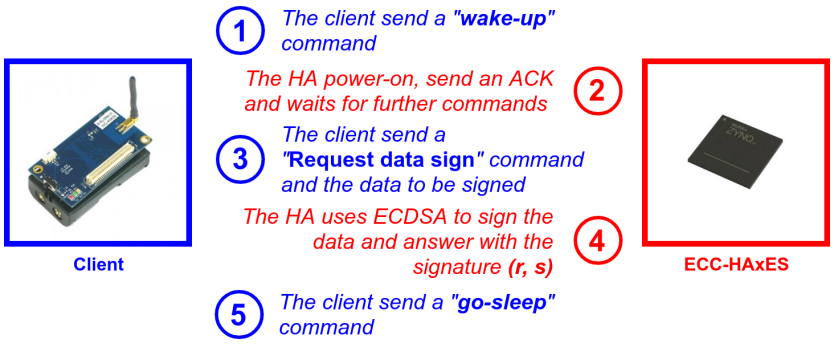


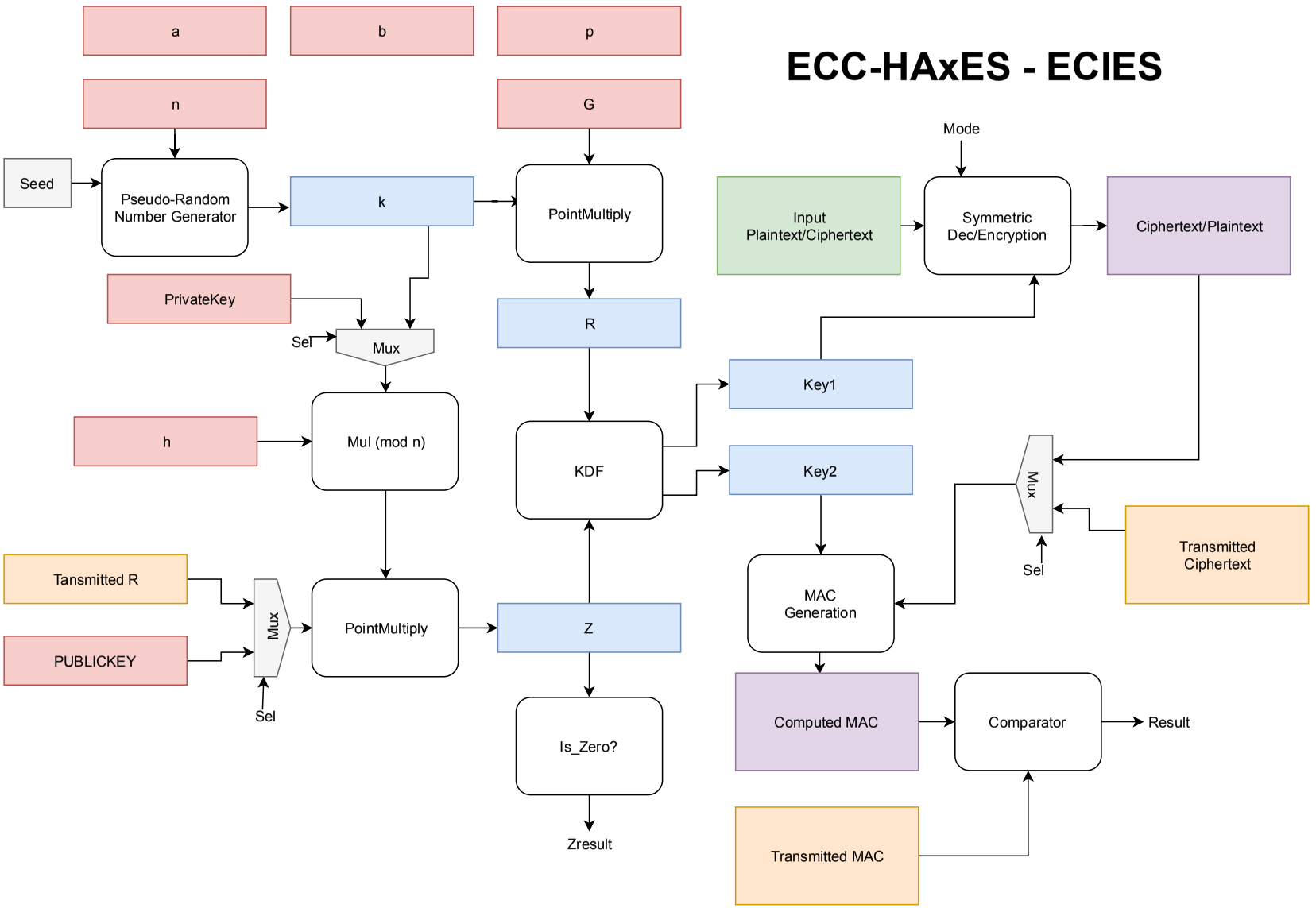
Figure 12.1: ECC-HAxES: Example scenario

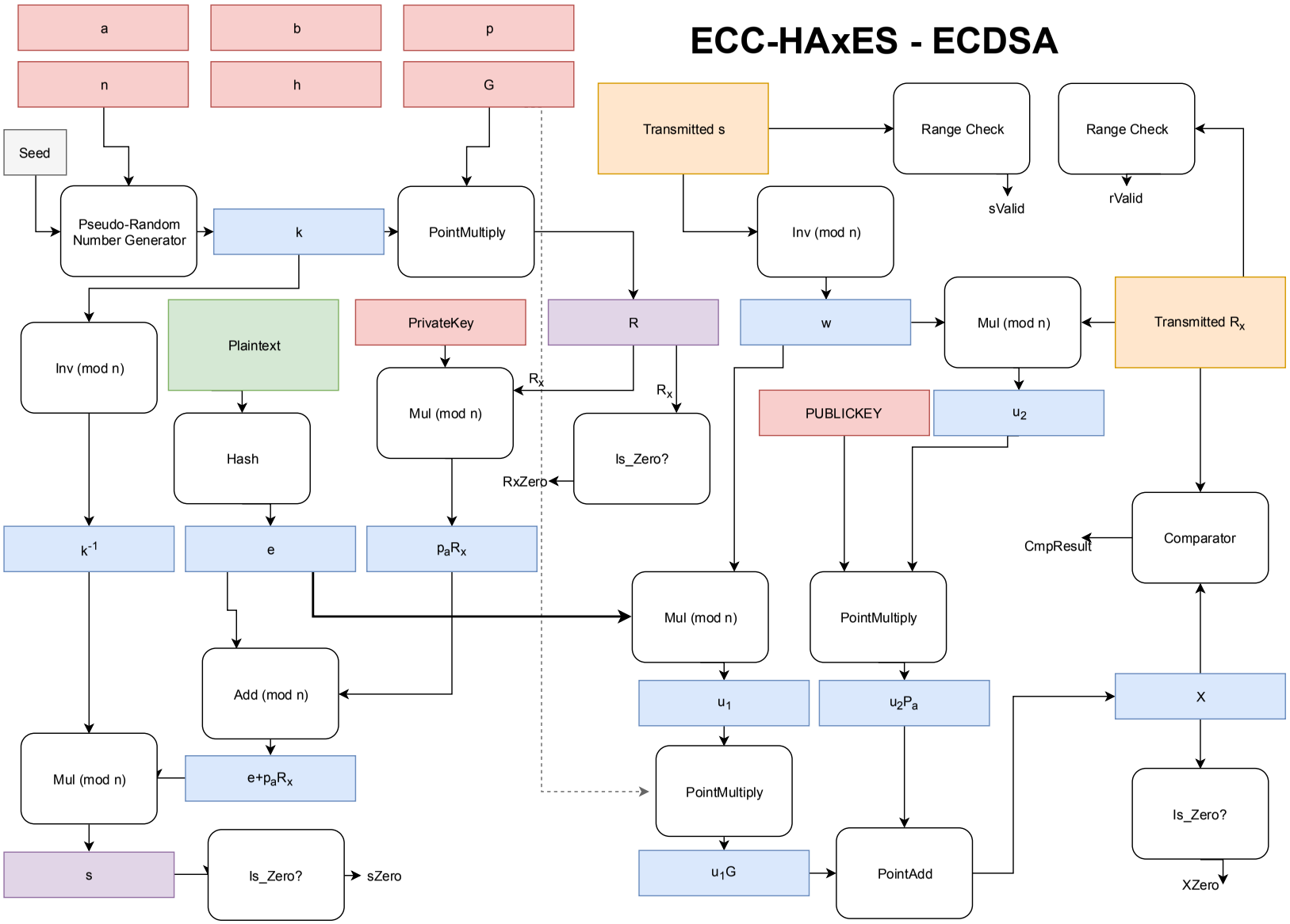
The client and the accelerator use a defined *communication protocol* to send *commands* and *data*. The first command a client issues to the accelerator is for *waking* it up from the stand-by state. Once the accelerator is ready, it sends an acknowledgement to the client and enters in a waiting state. At this point, the client can issue other commands, e.g., a request for signing a block of data, and sends the actual data to the accelerator. Then, the accelerator processes the data, sending the results to the client. Once the client has no more commands to issue, it sends a *sleep* command to let the accelerator go in the stand-by state.

Note that, the client private cryptographic keys are *never* transmitted through the communication channel: the accelerator manages them internally to help keeping them as safer (and secure) as possible. In that sense, the client can only issue a *initialization* command which causes the accelerator to generate a new set of keys.

Following the ECIES and ECDSA algorithms [53], below the high-level block diagram of ECC-HAxES are shown.

ECC-HAxES - ECIES




ECC-HAxES - ECDSA
a
b
p
n
h
G
Seed
Pseudo-Random Number Generator
k
PointMultiply
Inv (mod n)
Plaintext
PrivateKey
R
w
Mul (mod n)
Transmitted R_x
Transmitted s
Range Check
Range Check
sValid
rValid
Inv (mod n)
Hash
Mul (mod n)
R_x
R_x
PUBLICKEY
u_2
Is_Zero?
RxZero
k^{-1}
e
p_aR_x
CmpResult
Comparator
Mul (mod n)
PointMultiply
Add (mod n)
u_1
u_2P_a
X
Mul (mod n)
$e+p_aR_x$
PointMultiply
Is_Zero?
s
Is_Zero?
sZero
u_1G
PointAdd
XZero

12.3 Components

Starting from the block diagrams, a *bottom-up* approach has been adopted, i.e., the accelerator design started from the building block components, integrating them as the design phase proceeded.

The following sub-sections briefly describe these building blocks and how they work together to realize the final design. The following sections assume that the EC is defined over a *prime field* i.e., over $\mathbb{Z}/\mathbb{Z}_p$ with p prime.

12.3.1 Comparison of the design approaches

As stated in Section 12.1, the process for designing an accelerator architecture can be complex. In the recent years, new design techniques have been proposed to reduce the complexity of hardware design. In particular, a technique, called *High Level Synthesis* (HLS) [66] allow developers to use a software programming language to describe the required functions and, using a specific tool (the HL synthesizer), the hardware design (in the form of HDL code) is automatically created. In contrast to HLS, the *Register-Transfer Level* (RTL) design methodology could be used. An RTL architecture requires a deep knowledge of the digital design techniques and it is, in general, more time-consuming and error-prone. Despite to this disadvantages, it provides full control on the generated hardware, best quality, best performances and a simpler and cleaner design.

In order to evaluate, which design approach fits best the requirements listed in Section 12.1, both the HLS and the RTL approaches have been used and their results compared. Figure 12.2 shows the comparison of ECC-HAxES and its HLS counterpart.

	RTL Design	Naive HLS Design
Development Complexity	High	Medium/Low
Language and LoC written	VHDL, ~4300 lines	Xilinx HLS C++, ~300 lines
Final VHDLs LoC	~4300	~12700
Final Schematic Complexity	No change	Very High, unreadable
Result	Optimal	Unsynthetizable

Figure 12.2: ECC-HAxES: comparison of the RTL and the HLS implementations

Although the development time of the HLS took us a small development time, the results are orders of magnitude worse than the RTL architecture. However, improving the HLS architecture is possible by refining the HLS code using proper syntax, directives and optimizations, but, in general, this would cause the lost of its main advantage the reduced design time.

As conclusion, HLS could be used as first development step when the developers have a solid background in software development. After this step, however, a shift toward standard RTL design practices is advised, since the quality and performances of the resulting hardware is un-comparable.

Given the results above, in the presented research activities, the RTL design approach has been selected for the design of the accelerator. The rest of the

chapter describes the design steps and the validation of the inner components of the accelerator using the RTL approach.

12.3.2 Basic RTL

The first set of components are the basic building blocks for any digital electronics design (Figure 12.3).

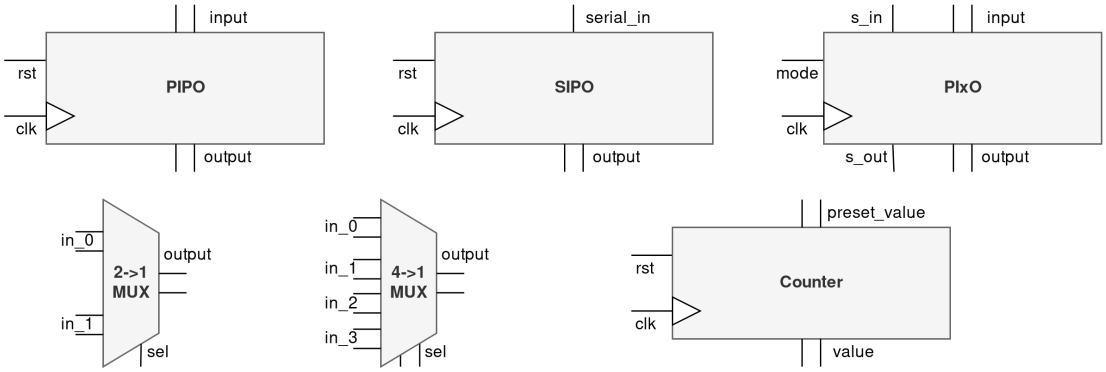


Figure 12.3: ECC-HAxES: Basic RTL components

- *Parallel-In, Parallel-Out Register* (PIPO register). This is the classical digital register, designed to have arbitrary size and an asynchronous *reset* pin. On every rising edge of the `clk`, data is read from *input* and stored and accessible by the *output* port.
- *Parallel-In, Parallel/Serial-Out Register* (PIxO register). This register allows the least significant bit to be set on the serial output pin. The size is arbitrary, data inside the register can be *shifted* in both the directions for an arbitrary number of bits. A *mode* pin is added to select the parallel load mode (`MODE=0`) or the shift mode (`MODE=1`). During shifts, input serial data is read from an external pin.
- *Serial-In, Parallel-Out Register* (SIPO register). Similar to the PIPO register, but it has only a 1-bit serial input instead of a parallel port.
- *Two-to-One Multiplexer*. A multiplexer with two arbitrary-sized input ports and one output port. The `sel` pin is used to select the input port to replicate on the output port.
- *Four-to-One Multiplexer*. Similar to the Two-to-One Multiplexer, but it has four input ports, a 2-bit selection port and one output port.
- *Up-Down Counter*. A arbitrary-sized counter which can be configured to count up or down. It also has an asynchronous reset pin and a port to pre-set a value in the counter state when a reset is issued.

12.3.3 Basic Arithmetic

This second set of components provide basic arithmetics for large integers in 2-complement representation (Figure 12.4).

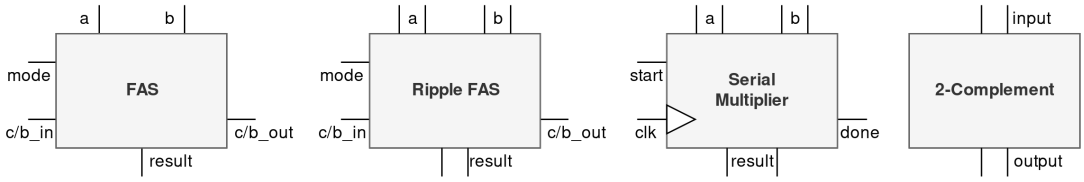


Figure 12.4: ECC-HAxES: Basic Arithmetics Components

- *Full Adder and Subtractor* (FAS). This component is a full combinatorial 1-bit *Full Adder* combined with a *Full Subtractor* to minimize area. The operation can be selected with a specific pin (addition: `OP=0`, subtraction: `OP=1`).
- *Ripple FAS*. This is a arbitrary-sized adder/subtractor obtained by multiple cascaded FAS. This adder (in respect of other solutions), when the size is large (e.g., 128 bits) ensure a reduced area occupation.
- *Serial Multiplier*. As for the RippleFAS, in order to minimize area, a *serial* architecture for the multiplier has been chosen. This component uses a RippleFAS as internal adder and, when the computation is started (`START=1`) it requires $O(n)$ clock cycles to complete the computation where n is the number ob bits of the operands. The result is a $2n$ bit value and it can be retrieved when `DONE=1`.
- *2-Complementer*. This components computes the 2-complement of the input value.

12.3.4 Modular Reduction

The *reduction modulo* p of an input value is one of the most important computation in a cryptography-related accelerator. Given an operand a and a number p (often a prime-number), it computes y where $y = a \mod p$ i.e., the remainder of a/p. While this operation is trivial in software, in hardware it requires a division which is not easy to perform efficiently in comparison with other operations. In literature there exists solutions to avoid divisions (e.g., the *Montgomery Reduction* [60]), but, after various tests performed on area occupation and performance, the modular reduction has been implemented with an optimized divisor which operates linearly with the size of the operands. The component is shown in Figure 12.5. The computation starts at the first rising edge of `clk` when `START=1` and the result is available when `DONE=1`. `DIV_ERROR` is `1` when the inner division operation failed (i.e., `P=0...0`).

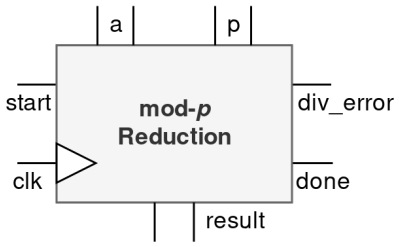


Figure 12.5: ECC-HAxES: Modular Reduction

12.3.5 Modular Arithmetics

After the definition of the basic arithmetic components and a modular reduction component, the modular arithmetics components required for the further EC computation have been designed. This section describes the modular addition, modular subtraction and the modular multiplication (Figure 12.6). The latter, in particular, is a core, performance-critical computation for the whole accelerator.

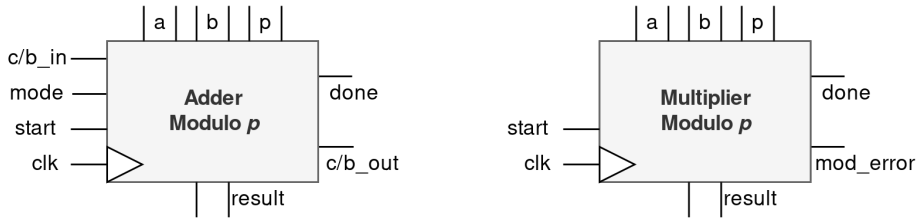


Figure 12.6: ECC-HAxES: Modular Addition/Subtraction and Multiplication

.

- *Modulo-p Adder and Subtractor.* The modular addition and subtraction are combined together to minimize area. It computes:

  1. $y = (a + b) \mod p$ when `OP=0`;
  2. $y = (a - b) \mod p$ when `OP=1`.

  This component uses a RippleFAS and the modular reduction to obtain the result. It assumes the input to be already reduced modulo p and, as other previously described components, the computation starts when `START=1` and the result is available when `DONE=1`.

- *Modulo-p Multiplier.* Despite the adder/subtractor which requires only a subtraction/addition of p if the operands are already reduced modulo p,

the multiplication component cannot make this assumption. This component combines the serial multiplier and the modular reduction to produce $y = ab \mod p$. `a`, `b`, `p` and the result have the same (configurable) bit length.

12.3.6 Modular Inversion

One of the most critical and complex component of the whole hardware accelerator is the *modular inversion* component. The modular inversion is the operation which computes y such that $ay = 1 \mod p$. y is guaranteed to exists *only* if p is a prime number. In modular arithmetics, this operation is usually preceding a multiplication for computing an *division-equivalent*. There exists various way to compute the modular inversion, for example, using the *Extended Euclidean Algorithm* (EEA). In [56] the authors analyze the common modular inversion computation algorithms for hardware-based solutions and concluded that the *RS* algorithm shows the best performance.

Using this result, in this thesis the RS algorithm has been adopted to implement the modular inversion hardware component (Figure 12.7).

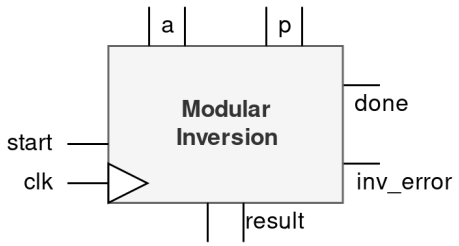


Figure 12.7: ECC-HAxES: Modular Inversion (RS)

12.3.7 EC Point Addition and Doubling

Given all the previous components, EC basic operations can be implemented i.e., the *Point Addition* and the *Point Doubling*. The point addition performs $R = P + Q$, while the point doubling computes $R = 2P$. P, Q and R are EC points expressed in carthesian (affine) coordinates (x, y) and encoded as $x||y$ (i.e., the concatenation of x and y). Since the two operations are the same, with the only difference in the computation of the *slope* s, a single component has been designed which combined both computations. (Figure 12.8). The slope is computed as $s = \frac{\Delta y}{\Delta x}$ in the point addition while $s = \frac{3x^2+a}{2y}$ in the point doubling (a is a EC curve parameter). Once s is computed, $R_x = s^2 - x_P - x_Q$ and $R_y = s(x_P - x_R) - y_P$.

If `OP=0`, the point addition $P + Q$ is performed. Instead, if `OP=1`, the $2P$ point doubling is performed. In both cases, the result is available when `DONE=1`. When `INF=1`, the result is the infinity point $\mathcal{O}$ and `RESULT` should be ignored.

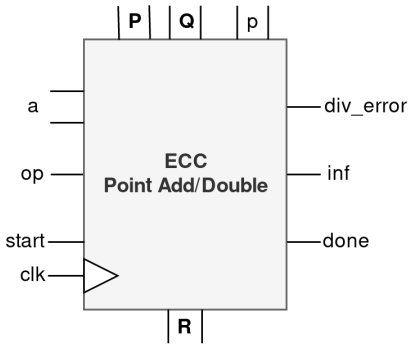


Figure 12.8: ECC-HAxES: EC Point Adder/Doubler

12.3.8 EC Multiplication

Another core component of the whole hardware accelerator is the *EC Point Multiplier*. It computes $R = kP$ with k a scalar value such that $k < p$ with p the prime number of the considered field. The component (Figure 12.9) uses the *Double-and-Add* algorithm using the point adder/doubler defined above.

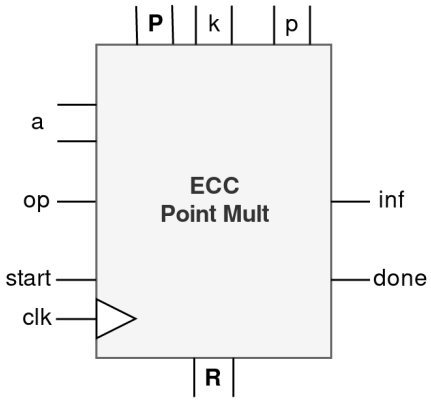


Figure 12.9: ECC-HAxES: EC Point Multiplier

12.3.9 Misc components

In addition to the components described above, some miscellaneous components have been developed to fill the gaps in the designs (Figure 12.10):

- *Ripple Comparator*. This component is a full-combinatorial comparator

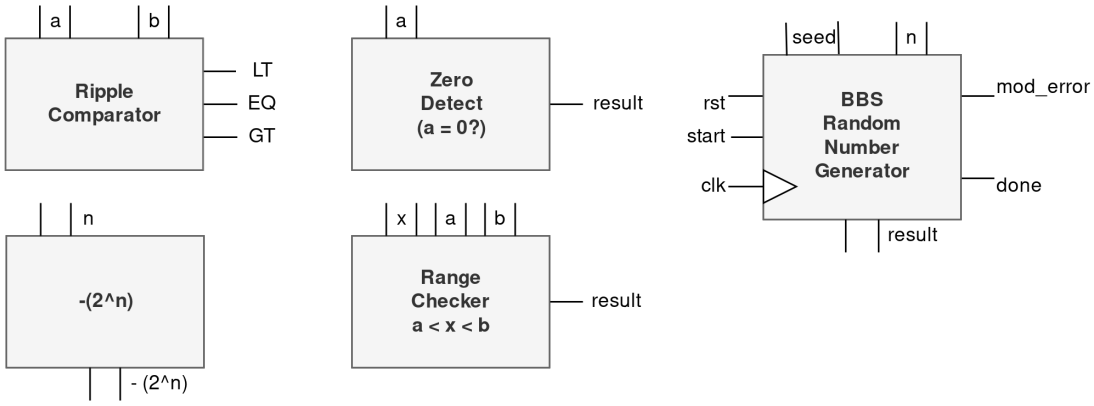


Figure 12.10: ECC-HAxES: Miscellaneous components

which adopts a *ripple* approach to save area. In particular, it is designed to compare bits from the most-significant to the least-significant.

- *Zero Detector*. This component uses the *XNOR reduce* port (i.e., performing the XNOR on all the bits of the operand) to check whether the input operand is zero (`OUTPUT=1`).
- *Range Checker*. This component combines two comparators to check whether the input operand x is in the range defined by a and b.
- $-(2^n)$ *generator*. This is a component used to quickly compute, given the exponent n, the corresponding power of 2 with its sign inverted.
- *Random Number Generator*. This component uses the *Blum-Blum-Shub* pseudo-random number generator algorithm to output a series of random numbers $\mod n$ using `SEED` as seed. A new random number generation is started by `START=1` and it is concluded when `DONE=1`. The internal state of the generator can be reset setting `RST=1`.

12.3.10 Top layers and work in progress

At the time of writing, not all the components required for performing the full hardware accelerator are designed. In particular, the components which are currently under design are the following:

- The *Key Derivation Function* (KDF) component, which generate a symmetric key from a point in the EC. The candidate function to perform such a task is *PBKDF2* [64].
- The *Symmetric Encryption* component, which encrypts or decrypt a plaintext. The candidates are *AES* [44] (block) or *ChaCha20* [46] (stream).

- The *Hash Function* component, which computes the cryptographic hash of a stream of bytes. The candidate is *SHA-256* [51].
- The *Message Authentication Function* component, which computes the message authentication code given a stream of bytes and a key. The candidates are *CBC-MAC* [81], *HMAC* [51] or *Poly1305* [65].

Once all the components will be designed, the design of the protocol-oriented components (*ECIES* and *ECDSA*) will take place in conjunction with the definition of a communication protocol for the accelerator and eventual adaptation components (e.g., a SPI/I2C slave component).

12.4 FPGA technology analysis

One question which arises when designing an hardware accelerator for embedded platform is the effective feasibility of attaching, using and powering an external board with the accelerator (as reconfigurable logic device or in ASIC).

To answer this question, the thesis research activities focused on investigating through the available FPGA platforms which provide a good amount of reconfigurable resources with a sustainable energy consumption, so that the WSN node could power the accelerator itself without the need of an external power supply unit.

Nowadays, all major FPGA vendors have a specific *low power* product branch, in particular, Lattice Semiconductor [100] and Microsemi [101]. At the time of writing, the FPGA platforms of both vendors consuming the lowest amount of power are the Lattice *iCE40 LP* FPGA family and the Microsemi *IGLOO2* FPGA family. Each product family offers different product with different performances, available area and power consumption, which is in the order of milli-Watts with a power supply voltage compatible with the one commonly provided by AA batteries (1.2 - 1.5 Volts).

12.5 Implementation

All the components described in the previous section have been implemented using the *VHDL* language. The implementation was performed using the GHDL synthesizer [61] and the Xilinx Vivado suite [62]. In particular, a top-level component consisting of a *Point Multiplier* and the required digital interfacing components have been implemented successfully.

The current implementation is available at the author's github page [63].

12.6 Validation & Results

The validations of the implemented components have been performed in three phases:

1. for each component, a *testbench* component has been created and simulated. The testbench is a non-synthesizable component used to provide a *black box*-like environment for testing a design (often called *unit under testing*, UUT) which can be *simulated* through a simulation engine for observing the behavior and results of the UUT.

2. a set of *top level* components have been created. These components aggregate a set of smaller components to provide an intermediate self-contained component useful for testing its inner components and their integration.

3. the top level components have been synthesized in order to configure them in a FPGA platform for validate the included set of components on a real environment.

The validation was performed using the GHDL simulator [61] and the Xilinx Vivado Suite [62]. Each component-level testbench was simulated and the components validated successfully across different tests. In order to provide a first top level, a component consisting of input SIPO registers, the ECC Point Multiplier component and output PISO registers to limit the amount of GPIO needed has been designed. We simulated successfully this top layer (Figure 12.11), implemented and synthetized it obtaining the area occupation results shown in Figure 12.12. The simulations and the implementation involve the kG computation using the *secp192k1* curve with various values of k.

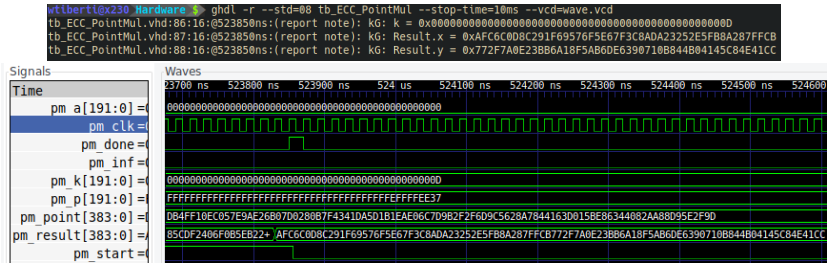
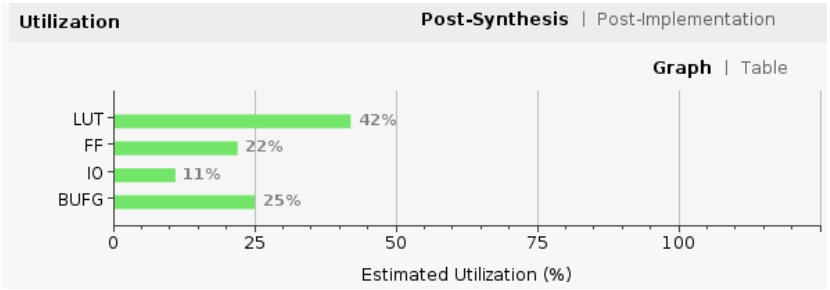


Figure 12.11: ECC-HAxES: Point Multiplication top layer results ($R = 13G$)

12.7 Future works

As discussed in Section 12.3.10, the current and future works are the design of the missing components of the accelerator and the development of the high-level protocol-oriented components. A validation phase on real FPGA platform and the performance evaluations will follow.

The work will conclude with the validation and deployment of the accelerator on a stand-alone board connected to a WSN mote, performance evaluation and comparison with software based approaches.




```
Report Cell Usage:
+------+------+------+
|      |Cell  |Count |
+------+------+------+
|1     |BUFG  |     8|
|2     |LUT1  |     1|
|3     |LUT2  |    24|
|4     |LUT3  |  1975|
|5     |LUT4  |   993|
|6     |LUT5  |  2407|
|7     |LUT6  |  3122|
|8     |MUXF7 |   579|
|9     |FDCE  |   193|
|10    |FDRE  |  7528|
|11    |IBUF  |     8|
|12    |OBUF  |     3|
+------+------+------+
```

Figure 12.12: ECC-HAxES: area occupation (target: Xilinx Zybo board)

Chapter 13

WSN Intrusion Detection System (*WIDS*)

Passive security functions described in the previous chapters ensure an high level of security for communication. However, they do not protect against attackers who already managed to get a foothold in the WSN (e.g., they captured and exploited a sub-set of the nodes). In order to address those threats, *active security* measures have to be employed. This chapter introduces *WIDS* [13], a lightweight *Intrusion Detection System* (IDS) specific for WSN. In the presented research activities, WIDS has been enhanced and implemented as a library (*TinyWIDS*) on top of TinyOS 2.x which other TinyOS-based applications can use to use its functionalities.

13.1 Motivation

As analyzed in Chapter 8, adopting an IDS in resource-constrained platforms is a non-trivial decision due to the impact it has on performances, throughput and memory. Among the two classes of IDS mentioned in Section 8, the choice for an IDS for WSN falls into the *Misuse-based* IDSs since they lighter than the *Anomaly-based* IDSs.

In literature, various approaches are proposed (e.g., [108][112] [104][105][106][107]) but none of them is able to provide a solution which takes into account the performance, the memory footprint, the energy consumption and the ability to detect newer attacks at the same time.

In [13], *WIDS*, a misuse-based IDS specific for WSN. As other misuse-based IDSs, WIDS is able to detect attacks with good accuracy and with a low impact on resources. Moreover, despite the limitation of other misuse-based IDSs, by using the *Weak Process Models* to describe attacks, WIDS is able to detect new attacks which shares a common behavior with existing attacks, hence outmatching the misuse-based IDSs limitation.

Thanks to this aspect, this research activities described in this thesis selected WIDS for further analysis and as base for the development of an IDS on top of the TinyOSv2 platform.

13.2 WIDS

The *Weak-Process-Model Intrusion Detection System* (WIDS) [13] is an IDS specifically designed to work in the high-constrained WSN platforms. WIDS is a *misuse-based* IDS which uses the *Weak Process Models* (WPMs) to detect when an attacker is targeting the WSN.

WPM are a special case of parametric *Hidden Markov Models*, where the hidden sequence of states of a WSN node is estimated through the *observable events* it produces. WIDS detects those events, collects the possible state sequences on the WPM and estimates the current state of the node. If such state is known to be a *dangerous* state, WIDS sends an alarm to the higher software layer or directly to the application.

WIDS models WPMs though a *graph* representation, where the nodes are the possible states and the edges are the possible *state transitions*. Each graph node has associated a list of the observables produced by a WSN node in that state. Some of the graph nodes represents the *dangerous* states, which are the states indicating an intruder currently attacking the WSN.

During the estimation, WIDS keeps a *threat score* value for each state sequence computed from the partial threat scores associated to every state transition. This value is used to measure the danger level reached at the current time.

Two are the types of dangerous states in WIDS: the *Low-Potential-Attack* (LPA) and the *High-Potential-Attack*. While in LPA states the threat score is evaluated before deciding whether send a notification or not, when HPA states are reached by any given state sequence, a notification of intrusion is sent, no matter the threat score.

An overview of a WPM and how WIDS works is shown in Figure 13.1.

In previous works on WIDS [13][10][11], WIDS has been designed to support the detection of various attacks, for examples:

1. Different classes of *Jamming attacks*
2. *Replay* and *Replay-Protection* attacks
3. *HELLO-Flood* attacks
4. *Workhole* and *Sinkhole* attacks

13.3 TinyWIDS

Previous works (e.g., [13]) validated WIDS through a series of simulation tests. The research activities of this thesis focused instead on providing a WIDS imple-

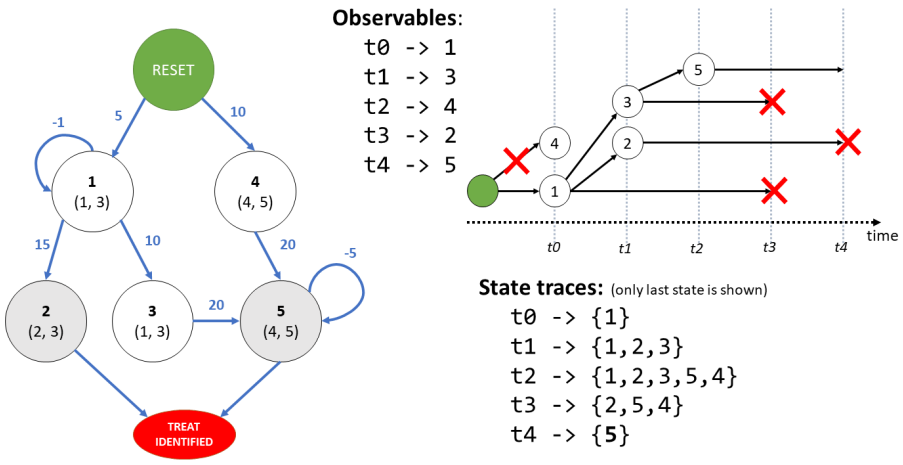


Figure 13.1: WIDS: a sample WPM (on the left) and a representation of WIDS state estimation (on the right)

mentation on real WSN node platform. As result, *TinyWIDS*, a first TinyOS-based implementation of WIDS, has been created.

13.3.1 Architecture

The basic architecture of TinyWIDS is shown in Figure 13.2.

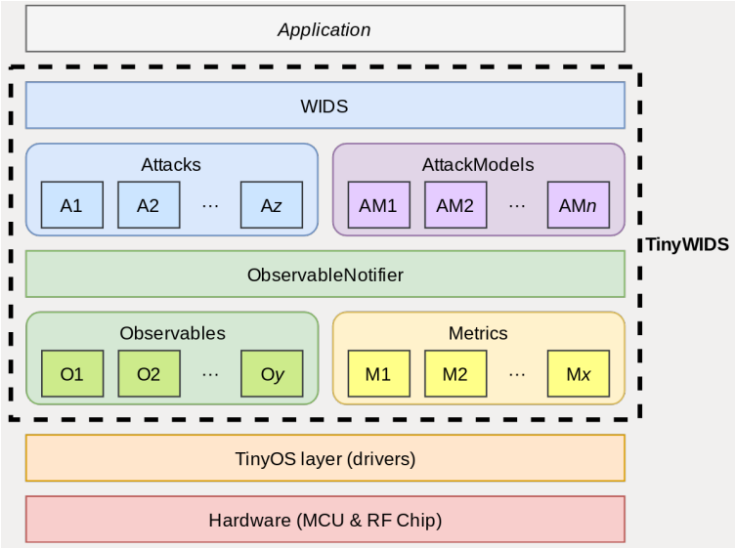


Figure 13.2: TinyWIDS Architecture

During the development of TinyWIDS, WIDS had the opportunity of being improved in different aspects. In particular, this thesis introduces the concept of *Metrics* i.e., components that keep track of a particular events or values from the hardware or from the driver layer (provided by TinyOS). Examples of what a *Metric* could be are the following:

- The number of successfully received frames (`RecvFramesMetric`)
- The number of frames failing the (hardware) CRC check (`CRCFailMetric`)
- The number of failed *Clear Channel assessments* (CCAs) (`CCAFailMetric`)
- The maximum RSSI value in the frames sent by neighbor nodes (`RSSIMetric`)

The *Metrics* allow us to have more flexibility in the definition of the WIDS *Observables*. In fact, in TinyWIDS, *Observable* are derived from a set of conditions (defined by the observable itself) on some metrics of interests. In this way, for example, a `ChannelBusyObservable` could be defined as:

$$\{RecvFramesMetric < x \vee CCAFailMetric > y \vee RSSIMetric > z\}$$

where x, y, z are numerical constants derived by the specific WSN context. The actual observables are created by a separated component called *Observable Notifier*. This component examines the metrics of interests and checks the conditions defined by the observables and, if any of the observables has met all of its conditions, it creates and insert such observable in a queue.

Another improvements brought by TinyWIDS is *Attacks* modeling. Describing an attack directly in a WPM is not an easy task neither a flexible solution. In TinyWIDS, we created a *JSON-based* syntax to describe an attack in terms of a graph and conditions. Then, a set of Python scripts are provided to build the NesC source code containing the definition of the WPM. In this way, unlike classical WIDS, in TinyWIDS it is possible to add, remove or change attack models easily.

Starting from the set of included attack models and a queue of observable, the *WIDS Engine* uses the WIDS mechanisms to walk the WPM and estimate the state of the node. When a HPA state is reached or the threat score reached a given threshold, the WIDS Engine creates a concrete *Attack* component, containing all the information on the current detected attack and all the audit/logging information retrieved during the state estimation. This component is then notified to upper layers (e.g., to the *Intrusion Reaction Logic* (IRL) or directly to the application) and stored for later analysis.

13.4 Validation and Results

TinyWIDS has been validated using the *Iris* WSN platforms and providing it with the attack models for a simple test attack (i.e., detecting when a frame with *sequence number* equal to 3) and for a more complex *Jamming Attack* based on

CRC and CCA failure rate. Tests have been conducted by putting a set *attacker* nodes trying to broadcast frames at maximum frequency, maximum power and ignoring any channel access policy. The conducted tests shown that TinyWIDS is able to detect such attacks with a good accuracy.

In respect of performance drawback, when using TinyWIDS as stand-alone components, the performance decreases linearly with the number of attack models embedded and the complexity of the observables involved. The memory overhead is, instead, linked to the complexity (in terms of nodes, edges, etc.) of the resulting attack WPMs.

In order to overcome such limitations, Section 15.3 describes a proposed integration of TinyWIDS with a MAMW.

13.5 Intrusion Reactions

In [13], it is not specified *what actions to undertake* when an intruder is detected, i.e., the so-called *Intrusion Reaction Logics*. This section presents one core enhancement of the presented research activities, the analysis of the possible intrusion scenarios and a set of possible automatic reactions aimed to both *locate* where the attacker is and to *reduce* its operativeness by confusing or isolating him/her.

13.5.1 Intrusion scenarios and reactions

Scenario 1: *Node Injection*

In this first scenario an attacker manages to silently intrude in the WSN some non-authorized nodes, with the aim of acquiring information, damage the WSN infrastructure or taking control of it via some exploits. The following assumptions are made:

- The WSN architecture (as well as topology, nodes position, protocols etc.) is known to the attacker
- The non-authorized nodes are indistinguishable copies of the original ones
- The WSN has an agent-based middleware (e.g., the one described in Section 15.1) , where:
  - Some agents gather data from node sensors
  - Some others check WSN status/parameters

After a notification from the IDS (Intrusion-detection system) a reconfiguration plan could be adopted to put the WSN into a safe condition, so that operators could be sent to physically remove non-authorized nodes (when possible). The reconfiguration plan can be one of the following:

1. **Buffer** plan. If the WSN contains only few non-authorized nodes, the idea is to create "isolation areas" around non-authorized nodes to exclude them temporarily from the WSN. An example is to power off (or just disable communication systems) every node in a circular (buffer) area around the non-authorized ones, with a radius of 1 or 2 hops. Using a mobile-agent MW, once the attacker and the injected nodes are located, a single agent could be injected to the WSN nodes in the buffer to power off/disable them.
2. **Jamming Buffer** plan. In this alternative, the nodes around send more and more random/un-useful data (jamming) to the non-authorized ones to confuse, slow down, overload or disable them while operators get on field for removal.

Scenario 2: *Node Tampering/Destruction*

In this scenario, the attacker performs physical tampering aimed to disable or destruct WSN nodes. The following assumptions are made:

- The WSN is deployed with a star- or cluster-tree topology, with one or more PAN coordinators (star centers or cluster heads)
- The data and software application running on the nodes is *critical*, hence powering-off nodes is not feasible solution.

When the attack has been detected, the WSN could be in a *post-attack* scenario, in which:

1. Some clusters/star have both PAN coordinators and device nodes in perfect conditions. In this case, no action needs to be taken.
2. Some clusters/star have a working PAN coordinator with one or more damaged device nodes. In this case, a **Change Coordinator** reaction could be undertaken. Since the cluster/star is under attack, the PAN coordinator and the sane device nodes are in danger. So, the sane device nodes in proximity of a second (sane) PAN coordinator, starts to pre-associate to it so that they can quickly perform a disassociation from the old coordinator when it eventually starts to not responding. In this way the functionality of the device nodes is preserved for longer.
3. Some clusters/star have a disabled PAN coordinator with one or more device nodes still in good conditions (*orphan nodes*). In this case, a **Coordinator re-election** reaction could be undertaken. Sane orphan nodes starts a temporary coordinator election among them (if possible), preferring the nodes with the best communication capabilities (e.g., number or reachable nodes) so that there is a better chance to restore most of the communications. Once the new temporary coordinator has been elected, it starts brodcasting its presence and accepting node association. This reaction require node-reprogramming capabilities, which can be provide by e.g., a MAMW.

4. Some clusters/stars are almost disabled. In this case, re-building a working cluster/star is not possible, so a possible reaction could be **Mute Cluster**, in which every sane node stops all the data transmissions to preserve energy, waiting for network operators to stop the attack before performing transmissions again. If the communications are also critical, a **WSN Remap** reaction could be undertaken as alternative. In this reaction, every remaining node floods the WSN signaling its presence. The global PAN coordinator (or the network operators) then, dynamically rebuild a valid cluster-tree/star WSN on the remaining nodes. As in the case of the *Coordinator re-election*, this reaction is deeply based on the node-reprogramming capabilities offered by the MAMWs.

13.6 Related publications

TinyWIDS concept description has been described by authors in [1] and its mechanisms illustrated in the poster session of the DATE 2019 University Booth [2]. Current version of TinyWIDS can be found in the GitHub repository located at [63].

Chapter 14

Blockchain-based security techniques for WSN

This chapter describes the issues and the proposed approaches in respect of WSN node data *tampering*. The chapter starts by analyzing the state-of-the-art (data) anti-tampering solutions for costrained-platforms; then this thesis describes a proposed solution involving the design of a *blockchain*-based anti-tampering mechanism which takes into account the WSN platform limits. Finally, some experimental results are reported, in terms of effectiveness and resource-footprint.

14.1 Anti-tampering techniques for resource- constrained devices

Apart from protecting WSN node communications via encryption and intrusion detection, often the overall security of the WSN resides in the nodes physical integrity as well as the integrity of the *data* they contain and manage. A WSN node with corrupted data can represent for an attacker a first *foothold* which can be used to further attack the target system. In fact, even subtlest data corruption (e.g., a *1-byte off buffer overflow*) in a innocuous buffer can provide attackers a way to corrupt and overwrite other memory areas, ultimately leading to *remote code execution* (RCE). While bigger and performance unconstrained platforms have different mechanisms to enforce data protection, WSN platforms in general have little or none protection in this sense.

This chapter, presents a recent research line which tries to address the data vulnerabilities in WSN monitoring applications before they happen. In particular, the research activities described in this thesis focused on providing a technique which *validate* the data incoming from WSN nodes before actually storing them. As positive side-effect, the WSN gained the ability to distinguish the WSN nodes which send *valid* data from those which send *invalid* data, allowing to provide a way to cut down the bad-behaving nodes similarly to an

intrusion detection but with increased effectiveness on the cases of WSN *node substitution* attacks.

WSN nodes are prone to both *physical* and *logical* tampering. In the former case, the WSN node is compromised with a physical modification (e.g., in its hardware components) to break, to misbehave or to cause damage around it. In the latter case, instead, the node is compromised in its *behaviour* (e.g., by changing its firmware) so that it is no longer behaving as when it was deployed.

In literature, various anti-tampering solutions have been proposed, also some specific to resource-constrained devices.

The physical tampering of a WSN node can be detected by using an interrupt-based technique through a low-cost trigger device as proposed in [87]. However, this technique requires the node to perform a checking procedures to reveal the tampering.

In [88] the authors propose the use of the Integrated Circuit metrics (IC-metrics) for the computation of metrics based on hardware and software characteristics. These metrics are used to generate cryptographic keys, so that they are dynamic and, in the case of tampering, they change.

In [89] Unpredictable Software-based Attestation Solution (USAS) is presented, which is an evolution of Software-based Attestation for embedded devices (SWATT) [90]. USAS is an algorithm for compromised nodes detection with the use of a hierarchical RC4 pseudo-random number generation, which starts from a Initiator node (I-node) and continues on multiple Follower nodes (F-nodes). For tampering detection the Base Station (BS) requests the I-node to perform a random challenge involving a checksum computation. The result of the computation is used to generate a series of new random challenge messages for F-nodes. The checksum results are checked by the BS for the compromised nodes' detection. Both USAS and SWAT introduce an overhead that can reduce the lifetime of the nodes.

In [91] the authors propose the use of Parameter Grouping for the detection of compromised node. Although the idea is valuable, there is a drawback: the whole test is conducted in a simulated environment where key parameters, like e.g., residual node energy, are provided by the simulator, whereas in a real environment these measured parameters could have been altered by the attacker.

In respect to the above mentioned works, this research activities described in this thesis focus on the *logical* anti-tampering and to provide a novel technique based on a lightweight blockchain to empower the WSN nodes with the ability to detect a compromised node into the network without introducing significant overhead. Nevertheless, it guarantees a good level protection of sensitive data and nodes operations and mission-critical operations.

14.2 *Lightweight Blockchain* (WSN-LBC) technique

The approach proposed by this thesis makes use of the *blockchain* technology, adapted to provide its benefits (e.g., immutable and verifiable data storage) also in the context of WSNs. This *lightweight* blockchain enabled to construct a anti-

tampering solution which makes compromised node detection easy and effective keeping, at the same time, all the blockchain-based data storage features.

In order to clarify the terminology adopted in this chapter, we adopt the following terms and meanings:

- *blockchain*, as immutable and verifiable data-structure, without any additional meaning from the *digital cash* context;
- *Ledger* as a term to indicate with a single, short and meaningful name the WSN node and the TinyOS component responsible of the blockchains management.

Considering a classical WSN monitoring application, consisting of a WSN PAN Coordinator node acting also as *sink* node, and a set of *device* nodes equipped with sensors. Our proposed approach defines a *Ledger* software component located in the sink node. The Ledger is responsible of the *message checking*, *blockchains management* and *data storage* tasks.

The message checking task is performed by means of a *Hash Function* component and a *Timestamp reader* component. The hash provides the Ledger and the user applications in the device nodes with a method to compute cryptographically-secure hashes. The Timestamp reader is instead used to compute time intervals. In Section 14.2.2 the message checking is thoroughly described.

The Ledger stores a set of multiple blockchains. These blockchains are ordered by the *reliability* of the data stored inside of them (e.g., the first contains high-reliable data while the last contains data with low reliability). The concept of *reliability* of WSN nodes and of their data is a central aspect of the proposed technique. We dynamically assign a *reliability* level (via *reliability points*) to WSN device nodes which determines the blockchain which the Ledger (located in the sink node) has to use to store node data. If a node have no enough reliability points, the sink node will eventually start to systematically refuse and discard its messages. Reliability points are increased or decreased depending on the result of the Ledger message checking task.

As already stated, data storage is performed by the Ledger using blockchains. The Ledger is configured with a variable number of blockchains. Due to the memory limitation of the WSN nodes, the *size* of each blockchain has to be a fixed (and usually small) value. Considering that sink nodes act also as bridges to traditional networking platforms *forwarding* (hence, *consuming*) the data gathered by the device nodes, a *time-windowing* approach with a circular-buffer mechanism has been implemented to replace the old data (which probably has already been forwarded) with the fresh incoming data. In this way, data can still be stored (and verified) into small and fixed-length blockchains.

14.2.1 Message Format

The message format adopted for communication is shown in Figure 14.1. It is built on top of the IEEE 802.15.4 MAC to increase data density and to provide

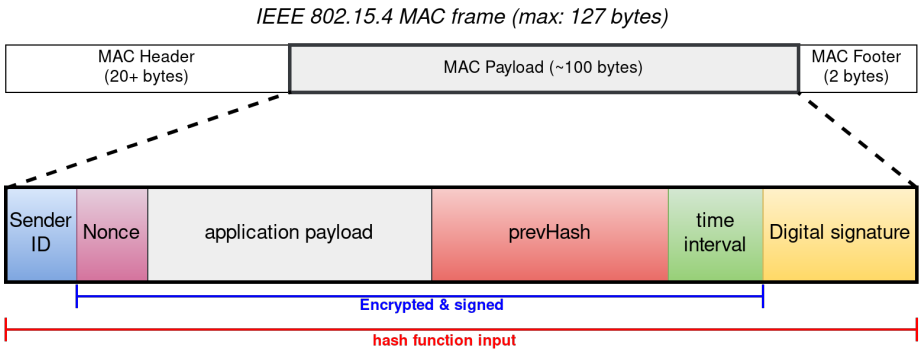


Figure 14.1: Proposed Lightweight Blockchain Technique: message format

the proposed features both directly to WSN applications and to higher-level protocols (e.g., ZigBee).

The message contains:

- the *ID* of the sender node;
- a *nonce* value;
- a *generic payload* field, freely usable by the application/higher layer;
- the *prevHash*;
- the *time interval* as computed by the sender;
- a cryptographic *digital signature* computed on the message.

The *sender ID* is the assigned to the sender WSN node during the programming step.

The *nonce* field is a pseudo-random values obtained by a cryptographically-secure pseudo-random number generator (CSPRNG) used to avoid *replay-attacks* by forcing each message to being different even with the same content.

The *application payload* field is an application-specific field, which can be used by applications (or higher software layers) to store the data of interest.

The *prevHash* is part of the main anti-tampering mechanism: the sender has to fill this field with the hash of its last un-encrypted message. The receiver (which should track the *prevHash* field, message after message) checks the field against its stored hash to determine the validity of the message. In this way, an attacker who wish to forge a valid message or perform a spoofing attack, has to know every previous message hash (down to the *first* message ever sent by the victim node) to successfully craft a spoofed message. Also, the attacker has to know the cryptographic key used both to encrypt the data and the key used to sign the message, or the message would be incorrectly deciphered or refused.

In order to provide an additional anti-tampering measure, the *time interval* field is used. In addition to the hash calculation, each sending node has to

keep track of the intervals between the messages it sends and fill this field by using one of its local free-running timers. On the other side, the receiver node does the same, keeping track of the received messages from each sender node (using its own timers). Even if computed with two not synchronized timers (i.e., the sender timer and the receiver timer), the time differences should be almost the same, and the receiver checks whether this happen to be. In this way, an attacker needs also to both synchronize with the victim node timer and has to know (according to this timing) when the last message has been sent. Even if this security measure is not strong as the previous one, it adds an additional difficulty level useful to deter attackers.

The *digital signature* is obtained by a digital signature protocol taking the message contents as input. The signature allows the receiver to authenticate the sender nodes and detect malicious attempts in modifying the message content.

All the fields described above are encrypted by the sender and decrypted by the receiver e.g., using AES with a key length of 128 bits and the CCM* mode of operation, in compliance with the IEEE 802.15.4 [16].

14.2.2 Message Checking

Upon the reception of a message, the Ledger:

1. Checks if the sender is considered reliable in terms of reliability points. The reliability points get increased (or kept constant) when the WSN note sends a valid message; they are decreased otherwise. If the sender is not reliable enough, its messages are discarded without any further processing.

2. Authenticates (by verifying a digital signature) and decrypts the contents of the received messages.

3. Performs the *Hash check*: the hash of the last decrypted legit message from the sending node is compared with the hash contained in the message. If they differs, the reliability points of the sender node are decreased.

4. Performs the *Time check*: the time elapsed from the last message from the sender node is computed both by the sender and by the Ledger in the sink node. The Ledger checks whether these time intervals have almost the same value (i.e., they differs up to a selected threshold). If not, the reliability points of the sender node are decreased.

5. After updating the reliability points, the Ledger decides whether the sender node should be promoted or degraded to a different blockchain. If the node's reliability points fall below a threshold properly selected, the Ledger denies the access to the blockchain storage. If not, the message is stored in the resulting blockchain.

The above described mechanism allows to provide both anti-tampering features to the application data (as any other blockchain-based mechanism) and to the nodes of the WSN themselves. In particular, the described approach allows

the detection of a WSN node that is tampered by an attacker, or injected into the WSN. In fact, actions like these cause the node to send messages which hardly pass the Ledger checks. Due to the low storage capacity of WSN nodes, limited data can be stored at the same time in a sink node.

An example of the message checking procedure and its results is shown in Figure 14.2:

- the WSN node 1 sends to the Coordinator (sink) a valid message, i.e., the hash and timing checks performed by the Ledger are successful, thus its messages are accepted. Subsequently, the reliability points of node 1 are increased or kept constant and its data is stored in a high-reliability blockchain.
- the WSN node 2 sends a message which contains a wrong `prevHash` and/or a bad time interval. So, the message is refused and the reliability of node 2 decreases, eventually causing the coordinator to start storing node 2's data into a blockchain of lower reliability;
- the WSN node 3 is classified by the Ledger as having already a very low reliability, hence when it sends a message which contains a wrong `prevHash` or a bad time interval, the message gets refused and, since node 3's reliability cannot decrease further, i.e., its messages cannot be trusted anymore, from now on its communications with the sink node will be discarded.

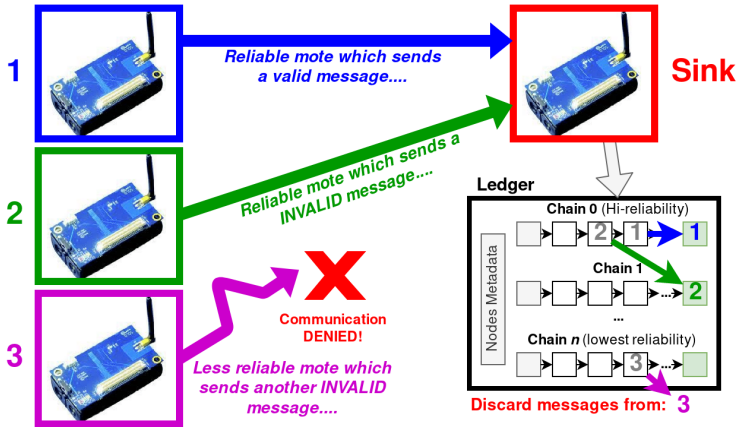


Figure 14.2: Proposed Lightweight Blockchain Technique: message checking

14.3 Implementation and Results

The proposed technique has been implemented using TinyOS. The Ledger is a new NesC component called `LedgerC`. The `LedgerC` component provide the

`LedgerI` interface, which can be used by the higher level software layer or by the application to access the proposed platform features. In this case, the famous *SHA 2 (256bit)* [51] cryptographic hash function has been selected, which is still considered extremely secure at the time of writing. The SHA256 digest computation has been re-implemented in a new separated component, named `SHA256C`. This component provides the `HashFunctionI` interface, used both by the Ledger and the application to compute hashes. Since this thesis considers TinyOS-based software applications, the `LocaltimeMicroC` component has been adopted as timestamp reader component. The `LocaltimeMicroC` is a TinyOS hardware-independent component, which can be used to retrieve the value of a free-running hardware timer with a granularity of a micro-second.

The UML diagram of the implementation is shown in Figure 14.3.

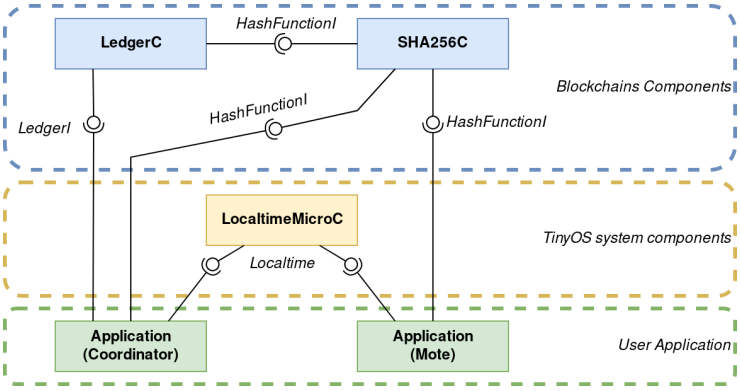


Figure 14.3: Proposed Lightweight Blockchain Technique: UML diagram

The proposed approach has been validated in a scenario consisting of four WSN nodes, three *device nodes* equipped with light sensors and one node acting as the sink node. Each device node periodically retrieves a selected number of samples from the light sensor. When a sufficient number of samples is collected, a new message is created (as described in Section 14.2.1) and sent to the sink node. The sink node, using the Ledger, performs the checks and provides as output (via serial port) the results and some additional logging information.

The validation results are shown in Figure 14.4 and Figure 14.5, where an example showing a node sending valid messages (Figure 14.4) and a tampered mote starting sending invalid messages (Figure 14.5). This latter behavior was obtained by simulating an attack in which the mote gets stolen, re-programmed and re-injected into the WSN.

The impact on performance and the storage footprint of the proposed approach is evaluated in terms of the following metrics: the fractional increment of the Flash and RAM memory occupation on device and sink motes, due to the introduction of our blockchain technique, and the average *Latency* introduced in the communications.

The metrics defined above have been evaluated through 18 different scenarios

```
[13873597] Message Incoming from node 1
Stored hash: c426412570055371eac13520fa29551ea9261fd05f9c9e82572ba2de400eea6d
Received hash: c426412570055371eac13520fa29551ea9261fd05f9c9e82572ba2de400eea6d
Checking block integrity...HASH OK!
Timestamp check OK!
Payload[32]: 4a0300000000000000000000000000000000000000000000000000000000000000
New block hash:c426412570055371eac13520fa29551ea9261fd05f9c9e82572ba2de400eea6d
[14237713] Message Incoming from node 2
Stored hash: ebdcc9799cb23f3d14c636b8d5e980e137db928d7d4177da26c5ae20f7865836
Received hash: ebdcc9799cb23f3d14c636b8d5e980e137db928d7d4177da26c5ae20f7865836
Checking block integrity...HASH OK!
Timestamp check OK!
Payload[32]: 4b0300000000000000000000000000000000000000000000000000000000000000
New block hash:ebdcc9799cb23f3d14c636b8d5e980e137db928d7d4177da26c5ae20f7865836
[14774344] Message Incoming from node 1
Stored hash: 2b04821676372d0b49dc40fda10bcf80525e97dde050e22f83cc3c61358717ea
Received hash: 2b04821676372d0b49dc40fda10bcf80525e97dde050e22f83cc3c61358717ea
Checking block integrity...HASH OK!
Timestamp check OK!
Payload[32]: 4d0300000000000000000000000000000000000000000000000000000000000000
New block hash:2b04821676372d0b49dc40fda10bcf80525e97dde050e22f83cc3c61358717ea
[15136585] Message Incoming from node 2
Stored hash: c1e16e7da737e44bd8e22d75d9bf969287cfb0ab4e967268e0e6c87e849e2cd9
Received hash: c1e16e7da737e44bd8e22d75d9bf969287cfb0ab4e967268e0e6c87e849e2cd9
Checking block integrity...HASH OK!
Timestamp check OK!
```

Figure 14.4: Validation: messages are sent, received and verified successfully

(first column of Figure 14.6), varying the following configurable parameters:

- The number of motes considered in the WSN (3, 5 or 10)
- The length (in term of number of stored data blocks) of each blockchain in the ledger (3 or 5)
- The number of different blockchains in the ledger (2, 3 or 4)

The tests have been conducted using as test platform the Memsic Iris motes [99].

In order to evaluate the metrics (and in particular the *latency*), the WSN motes have been programmed with a monitoring application consisting of motes exchanging messages with the sink mote. Each mote retrieves a fixed number of samples from the light sensors, starts a timer and sends the message to the other mote. Upon reception, the destination mote checks the message, stores it in its blockchains and sends a replay message back to the sender. When the replay message reaches the sender, this latter stops its timer and retrieves the round-trip time. The latency has been then computed by halving such value. Flash and RAM memory occupations are available directly by the TinyOS compilation toolchain when compiling the code.

We evaluated all scenarios, obtaining the results shown in Figure 14.6 for the sink motes. The resulting overheads are computed in respect of the same monitoring application compiled without using our proposed approach (the *baseline* application).

In Figure 14.6, the fifth column shows the overhead in Flash memory (i.e., executable code) with respect to the baseline application. Such a number is also expressed in term of percentage in the sixth column. The seventh and eighth

```
[14324437] Message Incoming from node 1
Stored hash: 0904877823b2eb87aa77cc0aaf54a28e96f9a63d7767d9e1ead7d4c70f8ca882
Received hash: 0904877823b2eb87aa77cc0aaf54a28e96f9a63d7767d9e1ead7d4c70f8ca882
Checking block integrity...HASH OK!
Timestamp check OK!
Payload[32]: 380300000000000000000000000000000000000000000000000000000000000000
New block hash:0904877823b2eb87aa77cc0aaf54a28e96f9a63d7767d9e1ead7d4c70f8ca882
[18172042] Message Incoming from node 1
Stored hash: 52cfedd099f03a7f4ff37d8fb0edf2567bc46453971b3b301cf16f3f638b570a
Received hash: 0000000000000000000000000000000000000000000000000000000000000000
Checking block integrity...WRONG HASH :(
Timestamp check OK!
Switching from chain 1 to 3
Payload[32]: 420300000000000000000000000000000000000000000000000000000000000000
New block hash:4188122200ffff01003f63010054180056f380300000000000000000000000000
[23955103] Message Incoming from node 1
Stored hash: a5e5ea87561120534719ef187a7cd5fee85404b1868202bedce388d6c646991d
Received hash: 9783139938e07a30fcac9b3f7332d88cabff68197fd47c7eff34c1099168504b
Checking block integrity...WRONG HASH :(
Timestamp check OK!
Switching from chain 3 to 255
[24855940] Message Incoming from node 1
Message refused (CHAIN DENIED)
[25756688] Message Incoming from node 1
Message refused (CHAIN DENIED)
```

Figure 14.5: Validation: simulation of an attacking mote

columns show instead the overhead in RAM memory (i.e., data) as relative value and percentage.

The results show that:

- the Flash memory occupation for sink motes has only a very small variation across the scenarios (less than 40 bytes). With respect to the baseline application, however, there is a 40% increase.
- the RAM occupation, instead, can grow very fast. The growth is due the use of RAM as data memory, so, it is sensible to the number of message storable in each blockchains and the number of blockchains.
- since rarely the WSN mote software uses a dynamic memory allocator, all the space needed is pre-allocated. This causes the number of motes in the WSN to be a direct factor in the RAM increase.
- it was not possible to test Scenarios 12, 14, 15, 16, 17 and 18 for latency evaluation due the amount of RAM required, which is more that the available in the test platform (8 KB).

Figure 14.7 shows the memory occupation overhead of a device mote in the case of the application running on the device motes. Those motes need only to construct a valid message to communicate with the sink mote and are not required to store data. The Flash memory overhead of device motes is less than $\sim 13\%$ in respect of the baseline application and there is a $\sim 43\%$ increase in RAM occupation. Notice that, since the device motes need only to be able to sample sensors, to construct a valid message and to send it to the sink mote, their memory occupation is independent from the metrics related to the memory occupation.

Scenario	Motes	Window size (# of blocks)	Block-Chains	Flash/ROM Overhead	%	RAM Overhead	%	Latency (ms)
1	3	3	2	4220	+39,51%	2248	+35,37%	172,935
2	3	3	3	4240	+39,69%	3025	+35,37%	172,965
3	3	3	4	4252	+39,81%	3802	+39,80%	194,63
4	3	5	2	4226	+39,56%	3136	+35,38%	173,01
5	3	5	3	4246	+39,75%	4357	+35,34%	172,82
6	3	5	4	4258	+39,86%	5578	+39,82%	194,72
7	5	3	2	4236	+39,66%	3288	+35,35%	172,855
8	5	3	3	4256	+39,84%	4583	+39,76%	194,42
9	5	3	4	4268	+39,96%	5878	+39,80%	194,625
10	5	5	2	4242	+39,71%	4768	+35,74%	174,775
11	5	5	3	4262	+39,90%	6803	+39,76%	194,41
12	5	5	4	4274	+40,01%	8838	---	N/A
13	10	3	2	4216	+39,47%	5888	+35,35%	172,875
14	10	3	3	4236	+39,66%	8478	---	N/A
15	10	3	4	4248	+39,77%	11068	---	N/A
16	10	5	2	4222	+39,52%	8848	---	N/A
17	10	5	3	4242	+39,71%	12918	---	N/A
18	10	5	4	4254	+39,82%	16988	---	N/A

Figure 14.6: Results: sink mote (containing the Ledger)

	Flash/ROM Overhead	RAM Overhead
WSN mote (device)	1474 bytes (+13.8%)	213 bytes (+43%)

Figure 14.7: Results: device motes

14.4 Related publications

The proposed anti-tampering technique has been submitted to the ITASEC 2020 Italian Cyber-Security conference [4].

Chapter 15

Agilla Evolution

This Chapter moves the focus on the base software environment chosen to provide the security functionalities proposed in the previous chapters: the *Mobile Agent Middleware* (MAMW). Given the unique features, *Agilla* MW has been selected as baseline for further enhancements.

The following sections describe the progress introduced by the presented research activities to Agilla and its new features. Finally, this thesis proposes a first design of a complete MAMW-based security solution which combines the MAMW with the cryptographic schemes, the intrusion detection and all the other proposed solutions.

15.1 Towards *Agilla2*

15.1.1 Motivations and Contributions

In WSN, as well as in other embedded systems, the software development, management, distribution and update is not as simple as in other computing platforms. The WSN software is usually developed using cross-compilation techniques due to the unfeasibility of hosting a complete development environment directly on the target platform. This issue causes the necessity for developers to manually compile (i.e., *cross-compile*) and transfer the compiled software to the target platform by means of wired links (e.g., serial ports) and protocols (e.g., RS232/UART, JTAG, USB) Moreover, embedded applications often rely on one or more software layers, namely operating systems, software wrappers, *bootloaders*, etc., that allow the execution of one application on different hardware platforms. The tight relationship between the embedded software application and the lower software layers makes the compatibility with new software versions a non-trivial issue. Even a small change to lower layers affects high-level functionality, eventually causing the application to stop working as intended. Often developers have to re-design and/or re-write the application to restore the full compatibility with the updated lower software layers.

This is the case of *Agilla* MW, originally developed on top of TinyOS 1.x. Indeed, during the thesis research activities in WSN and in mobile agent middleware domain, Agilla has been successfully adopted in many contexts (e.g., see Section 16.2) However, the new release of TinyOS (v2.x) caused Agilla and every other TinyOS 1.x application to stop being compilable on common WSN hardware platforms.

In this part of this thesis, Agilla compatibility with TinyOS has been restored and new features from the new version of TinyOS has been added. To do so, the thesis research activities addressed the problem of porting Agilla from TinyOS 1.x to TinyOS 2.x. Although a simple porting activity might be not of interest in the research community, in the particular context considered (i.e WSN, limited resources, energy-constraints, no low-level software support, hardware heterogeneity), porting activities pose non-trivial issues for which in the current state of art in software maintainability for embedded systems there is no general technique. Such activity has been called "porting", since the reimplementation of the software application aims primarily to fix the features and functionalities compromised by the changes occurred in the lower software layers.

During the porting activities, Agilla has been enhanced by adding features or improving the ones already present. Such operations required a deep knowledge of Agilla source code and its inner mechanisms in order to improve on-the-fly parts of it to better suite the new requirements and to gain various advantages, in term of software quality (e.g., performance, energy consumption, etc.). This knowledge has been used to model the architecture of Agilla and its interdependency with TinyOS 1.x (using UML and graphs). The obtained models are not provided by its official documentation and are useful in the future Agilla maintenance.

The followed porting methodology can be applied to different embedded applications, so it has been generalized. The result is a *model-based* approach that helps to keep under control the complexity of the *porting* of embedded applications.

15.1.2 Model-based Porting

The knowledge of both the starting application and the target platform is *vital* to obtain a good quality *porting* of a software application. In this sense, this thesis proposes a *Model-based* approach which focuses on retrieving one or more software models from the available source code and documentation to acquire a deep knowledge of the application while having, at the same time, a solid environment for code modification and feature additions.

The overall approach is sketched in Figure 15.1.

In the figure, the yellow boxes represent artifacts (e.g., code, models, etc.) while the rounded blue boxes represent the processing steps.

As first activity, the available source code of the target application is retrieved and reverse-engineered in order to obtain an architectural model describing the software architecture of the software. Standard UML is used as

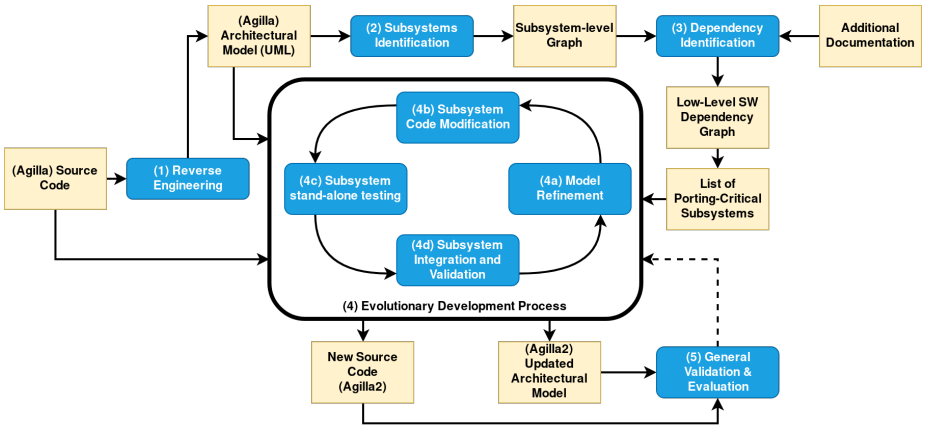


Figure 15.1: Model-based porting overview

architecture description language to provide the architectural model required in the following steps. This step is not completely automated since there is not any toolkit for retrieving full-featured UML diagrams that works directly on the starting programming language. More details on the Agilla reverse engineering and on the derived UML model are reported later in this section.

The second step subdivides the models obtained from the previous step so that all the components are grouped by the higher level functionality they aim to provide. These components groups have been called *subsystems*. This step is very important in bigger applications, since helps developers to reduce the analysis surface and focus on functionality-oriented porting process. After this step, a subsystem-level graph is created.

Using the graph retrieved in the second step, the third step filters the graph such that it is clear which subsystem has a strict coupling with any of the lower-level components, thus, identifying the dependency of the target application on the (different) lower-level software layer. In the Agilla case, this step let us to identify the *porting-critical subsystems* of Agilla, i.e., the subsystems containing components that have a major impact on the porting procedures due the higher coupling with TinyOS. This step considers the other sources of documentation available, both for the target application and for the lower-level software. In particular, the available documentation of both Agilla and the *TinyOS Enhancement Proposal*s (TEP) documents have been analyzed, while focusing on those describing the differences between TinyOS 1.x and TinyOS 2.x. The dependency analysis, the analysis of the additional documentation and the retrieval of the *porting-critical subsystems* are described in Section 15.1.2.

After the third step, the *Evolutionary Development Process* is then executed. This process takes as input the Agilla source code, its architectural model and the list of Agilla *porting-critical* subsystems to produce the architectural model and the source code of new version of the target application, in this case, *Agilla2*.

The process consists of multiple iterations of modeling (*Subsystem Model Refinement* steps), coding (*Subsystem Code Modification* steps), testing (*Subsystem stand-alone Testing* steps) and integration/validation phases. Modeling, coding, testing, integration and validation steps are interleaved rather than separated, with rapid feedback across activities. The process is repeated for each subsystem and ends when the source-code and the architectural model for the subsystems considered are validated and integrated.

Finally, a general application-wise validation step is conducted. This step ensures that the ported application can be used with all its features in place of the old version. If the validation is successful, the application is evaluated in terms of performance gain/loss, memory occupation or other relevant metrics. If not, the Evolutionary Development Process is started over to refine the models and correct the source code.

In particular, to evaluate the porting results, the improvements and the software quality, a set of metrics have been defined and measured on Agilla2.

Agilla Reverse-Engineering

The first step in the model-based approach starts with the analysis of the available source-code of the application, in order to perform the *reverse-engineering* and obtain an abstract and possibly standardized architectural model; the model can help developers to comprehend the application features, behaviour and, most important, give a dynamic support *tool* for the following steps of the proposed approach. Also, a model can be useful in future, for maintainability of the application and to easily provide further improvements.

Using the UML notation it is possible to describe large systems but, in Agilla case, the output diagram could be not so lean and easy to understand due the high number of components and interconnections (i.e., interfaces provided or used). The proposed solution is to focus on the description not on every single component, but on every *component hierarchy* of the application (as shown in Figure 15.2). In this way, it was possible to obtain diagrams at different levels of granularity e.g., from *system-level* diagrams to *component level* diagrams.

The diagrams have been created by using TinyOS `nesdoc` utility and the *MagicDraw* software application.

Agilla Models

The UML Component Diagram obtained in the first step is complete and offers a description of Agilla down to the components and hierarchies relations. However, this fine-grained level of information poses a non-trivial challenge on the developers which, in the next steps, have to work on it. In the proposed approach, the second step (*Subsystem Analysis*) tries to provide a solution: the creation an additional, coarse-grained model from the starting UML Component Diagram to highlight the logical group of cohesive components (and components hierarchies) considering the *features* they mean to provide in the overall application. Such groups of components are the *subsystems* of Agilla (Figure 15.3).

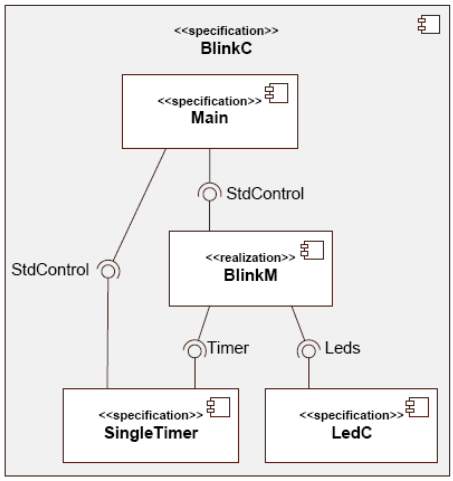


Figure 15.2: Model-based porting: Component hierarchies

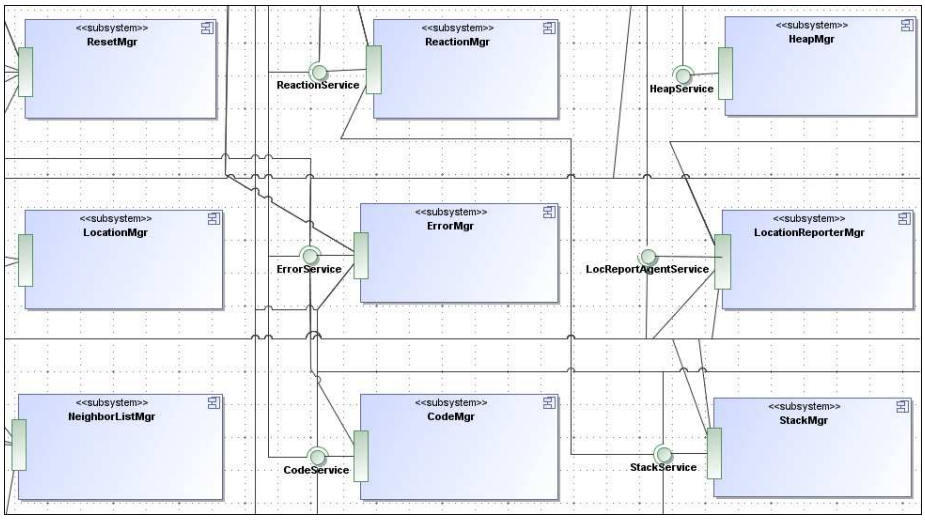


Figure 15.3: Model-based porting: Component hierarchies

Agilla and TinyOS Dependency

In order to proceed with the next step, the diagrams obtained from the previous steps have been investigated to separate application-related components from low-level basic components and reduce the analysis surface. For example, in

Agilla, components like *LedsC* or *SounderC* (core TinyOS components used to control the LEDs and the buzzer) can be pruned from the analysis since their inner mechanisms are out of the scope of the porting. Instead, the analysis focuses on Agilla-only components, which may need to be reworked to obtain a working porting, An example is the *OPSleepM* component: it is the component responsible of the implementation of the agent macro-instruction `sleep`, used to introduce a user-defined delay in the agent code. Since it internally uses some of the TinyOS interfaces which have been changed in the 2.x version, this component is more prone to be changed during the porting procedure in order to restore its functionality.

In this step, any additional available documentation on the target application and on the changes that affected the lower-level software can provide valuable information to help developers both to analyze the dependency of the target application and, later on, to identify which components need to be reworked. In the Agilla case, the proposed changes and those then introduced in TinyOS from version to version are documented in a collection of documents called *TinyOS Enhancement Proposals* (TEPs) [114] available in every distribution of TinyOS. By reading the TEPs, it was possible to identify and track almost all the changes introduced in TinyOS 2.x, thus identifying the impact on Agilla affected subsystems.

After the dependency analysis, the models (both the UML Component Diagram and the Component Graph) were refined again, pruning out all the unused TinyOS components and adding additional description on the Agilla components to include information (from the TEPs) useful for the next steps. The remaining subsystems and components are, respectively, the *Critical Subsystems* and *Crtitical Components*.

Agilla Evolutionary Development Process

The *Evolutionary Development Process* consists of 4 steps (Figure 15.1). The first step is the *Model refinement*: focusing on the subsystem components, this step refines and enhances the information quality and quantity on the UML Component Diagram and Component Graph models. For example, added information on where the components are located, which algorithms and data structure use to achieve their functionalities and so on. As a consequence, the refined subsystem models contain information that is useful to the developers in the following step of source code modification. Moreover, as said previously, in this step developers can decide to improve some aspects of the architecture of the subsystems.

Once a refined model of the target subsystem is ready, the actual source code modification can take place (the *Code Modification Step*). Using the original source code and the refined model as supports, the original source code is analyzed and reworked to fit the requirements of the target platform.

In order to ensure that each reworked subsystem is working, even before the validation, this step creates a *stand-alone testing* environments (*testbeds*) for testing the subsystem without compiling the whole application. This step helps

developer to quick check and fix problems from the previous step, potentially reducing the number of issues that may appear after the integration of the reworked subsystem after the integration in the application. Also, by testing only the target subsystem in its testbed, developers can focus on the subsystem functionalities, reducing the quantity of code to review/test and the time required to perform such operations. The creation of the testbeds for Agilla consisted in creating new, small, TinyOS 2.x applications which include only the target subsystem and some basic boot code.

The final step of each iteration in the process consist in integrating the subsystem in the application. With a successful integration, the target subsystem is connected back as drop-in replacement into the application. Since the interfaces used and provided by the target subsystem may have been modified, in this phase developers can discover and list inter-subsystem problems which can be handled in a subsequent iteration of the process of the same subsystem.

Agilla2 Validation

After the execution of the Evolutionary Development Process, a runnable version of the ported application is ready. The final step of our model-based approach consists in an overall validation which ensures that the new version of the application can actually replace the old version. The ported Agilla (*Agilla2*) has been compiled by exploiting the TinyOS *make* system, which automatically launch the current cross-compiling tools to produce the final programmable binary file. A first validation check can be performed just after compilation, by analyzing the compilation results against the capabilities of the target platform. In this sense, the compiled Agilla2 storage requirements (obtained directly from the compilation phase) was compared against the memory storage available in the target mote hardware (MicaZ). The obtained Agilla2 have an occupation of 56 KB of ROM (code) and 3.6 KB of RAM (data). These values are compatible with the MicaZ mote storage limits (128 KB of ROM and 4 KB of RAM).

After a correct compilation of Agilla2, was the turn of the *AgillaAgentInjector*. The *AgillaAgentInjector* Java GUI application is used to connect to motes and inject agents into them. The compilation was successful and in Figure 15.4 the two compilation outputs are shown.

The final validation step has been to check the runtime correctness of Agilla2. The goal has been to check if it was possible to run an agent-based application correctly on Agilla2, using all the available features, such as sensor readings, communications, and migration of agents. Among the available agent-based application in Agilla, *Oscilloscope* is one of the most meaningful: it offers a oscilloscope-like visualization of the sensor data retrieved from all the mote in the WSN against time, visualizing a waveform. In order to perform the validation, Agilla2 has been installed on the *MicaZ* and then the proper agent has been injected through the *AgentInjector*, which started to successfully retrieve data and forward them to the PC. On the PC, the *AgentInjector* interface and its Oscilloscope component has shown, as expected, a waveform related to collected data. This scenario has been tested both on PC and on an Android-based

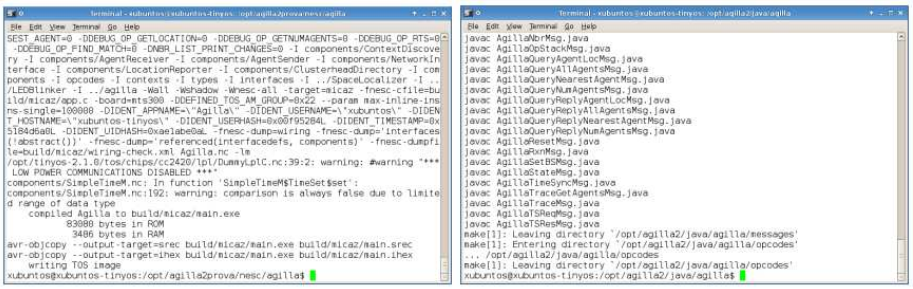


Figure 15.4: Agilla2: successful validation

device, with successful results (Figure 15.5).

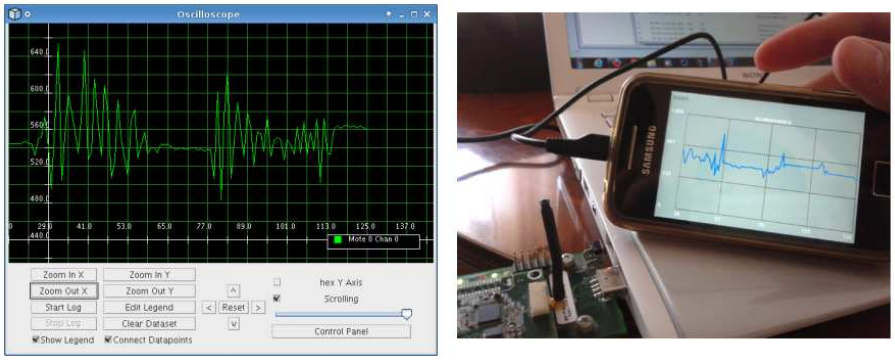


Figure 15.5: Agilla2: successful Oscilloscope validation

Finally, after the successful validation, Agilla2 has been released and it can be found in [95].

15.1.3 Agilla2 performance & quality analysis

After-Porting analysis

This section discusses and compare the ported version of Agilla and its original version in terms of *software quality*, *performances* and *maintainability*. Also, from an higher point of view, the objective has been to determine whether the *cost* of the model-based porting approach (e.g., in terms of men/hours spent to obtain a working Agilla2) is worth the efforts. In order to estimate the *cost* of performing the porting, the differences in terms of performance and memory footprint between the original Agilla (on top of TinyOS 1.x) and Agilla2 (on top of TinyOS 2.x), we have defined a *set of metrics* (Table 15.6).

Then, the original Agilla and Agilla2 source code have been instrumented, compiled and investigated to retrieve such metrics.

Metrics List		
Metric name	**Meaning**	**How it is evaluated**
Agilla ROM footprint (ROM)	The space occupation (in bytes) of the binary image of Agilla when programmed into the node	The ROM occupation is retrieved by the NesC compiler and showed in the compilation output log
Agilla RAM footprint (RAM)	The estimated runtime space occupation (in bytes) of the data allocated by Agilla when programmed into the node	The estimated RAM occupation is retrieved by the NesC compiler and showed in the compilation output log
nesC Lines of Code (nesCLoC)	The total number of line of NesC code in Agilla components, (without the TinyOS components)	The LoC are computed through the standard `wc -l` UNIX command on all the Agilla components
Resulting C Lines of Code (CLoC)	The total number of line of C code generated by the nesC trans-compiler from Agilla (including TinyOS components)	The LoC are computed through the standard `wc -l` UNIX command on the nesC generated file `app.c`
Quality of Documentation (Doc)	The quality and the quantity of documentation available both inside and outside the source code	Source code comments, Specifications and models are analyzed in order to assess the quality of the documentation. Moreover the comment ratio [**?**] and the artifact ratio (percentage of artifacts per component) are also reported

Figure 15.6: Adopted Matrics for Porting Evaluation

- Source code lines (NesC) are retrieved by a summing the lines of each source file.

- The resulting C source code lines can be retrieved by counting the lines of the `app.c` file, which is generated by the NesC trans-compiler upon compilation. Although a raw measure of the source code lines has no strong meaning, it is useful to consider the ratio between the two metrics:

$$\frac{CLoC}{nesCLoC}$$

  The resulting value is higher when few lines of NesC code generate a high number of C source code lines. This is good indication of the *expressivity* of the NesC source code.

- The storage occupation (both RAM[1] and ROM storage) are computed directly upon a successful compilation of Agilla/Agilla2. As in any embedded system, the desired value for those two metrics is *as little as possible*.

- Finally, the quality of the documentation in the Agilla/Agilla2 source code was analyzed. Two metrics have been used: the *comment-ratio* and the *artifact ratio* which definitions can be found in [96]. Apart from the source code, this step consider also the documentation created during the porting operations.

Table 15.7 shows the values retrieved for the defined metrics. These results show a general improvement of Agilla2 source code and documentation over the original Agilla.

[1]RAM occupation takes into account only statically allocated data. Dynamically allocated data is not included, although it can be neglected since Agilla use only pre-allocated data

Results		
Metric name	**TinyOS 1.x + Agilla**	**TinyOS 2.x + Agilla2**
ROM [bytes]	54736	55944 (~+2.2%)
RAM [bytes]	3191	3640 (~+14%)
nesCLoC [number]	19776	21563 (~+9%)
CLoC [number]	40285	61402 (~+52%)
Expressivity [ratio]	2.037	2.84 (~+39%)
Doc	Source-code comments	Comments, TEP, **Graphs & Diagrams**

Figure 15.7: Metrics evaluation results (`micaz` target)

Instruction-level performance analysis

An additional performance evaluation was performed on Agilla2 *Instruction Set Architecture* (ISA). Agilla2 has been instrumented in its core execution engine (the `AgillaEngine` component) in order to retrieve the execution time of each executed instructions. The latency of every instruction has been retrieved by crafting ad-hoc agents to cover the entire ISA. The results (expressed in micro-seconds) are shown in Figure 15.8. In the figure it is possible to observe the

INSTRUCT	MAX	MIN	AVG	MODE	INSTRUCT	MAX	MIN	AVG	MODE	INSTRUCT	MAX	MIN	AVG	MODE
rand	352	249	263,61	249	add	360	129	154,20	129	pushrt	323	91	118,56	93
lnot	108	108	108,00	108	cgte	342	109	186,67	109	cisnbr	102	95	98,50	95
dist	264	144	184,67	144	swap	373	142	219,00	142	land	526	499	508,00	499
pusht	691	89	118,43	90	cpush	319	88	115,33	89	rdp	481	391	422,17	423
regrxn	301	218	230,25	219	setvars	118	115	116,33	115	lor	129	129	129,00	129
inv	108	107	107,17	107	mul	155	129	141,83	129	pushloc	264	177	184,67	177
sense	10317	9037	9453,33	9037	numnbrs	90	89	89,17	89	rrdp	4078205	8558	1963069,33	
inp	1062	319	571,83	319	pushn	175	170	172,33	170	endrxn	4607556	4607207	4607320,50	4607207
pop	478	79	125,91	81	setvar	467	96	143,96	98	cneq	111	110	110,42	110
clte	343	111	188,67		deregrxn	538	306	364,25	306	clearvar	331	90	114,60	94
pushc	690	89	104,02	90	ceqtype	110	108	109,00	109	randnbr	282	281	281,33	281
sleep	230677	824	132126,32	114768	rinp	4044665	6481	1288670,67		getnbr	90	89	89,11	89
in	626	625	625,67	626	routg	230637	230548	230596,67		aid	93	90	91,17	90
wmove	580996	577618	579644,33		jumps	460	460	460,00	460	div	159	159	159,00	159
pushcl	771	171	333,56	172	rjumpc	91	72	75,15	75	dec	185	126	155,33	126
sclone	34256	23965	27512,00		wclone	20356	18046	18990,00		cistype	109	109	109,00	109
rrdpg	230871	230871	230871,00	230871	clt	111	110	110,58	111	clear	453	198	227,91	201
inc	125	110	117,67	125	putled	340	99	116,50	100	out	601	246	358,08	247
loc	526	157	220,63	157	ceq	111	110	110,67	111	and	113	110	111,50	110
addr	90	88	89,13	89	not	339	109	185,67	109	or	339	107	146,33	107
getvars	350	112	180,83	112	getvar	327	95	111,33	96	rjump	78	74	76,17	78
rout	14710	10013	12380,67		jumpc	321	90	112,27	90	and	113	110	111,50	110
cpull	89	88	88,33	88	copy	122	121	121,33	121					
smove	587445	578445	584145,33		cgt	110	108	109,00						
rd	670	424	546,67	424	mod	156	156	156,00	156					

Figure 15.8: Agilla2 instruction execution times (μs)

slowest instructions (highlighted in cyan). Apart from the `sleep` instruction (which is meant to have such a latency) and the `in/rd` instructions (which are *blocking* instructions), those *slow* instructions can be grouped into:

- Remote Tuplespace management (`rout, rinp, routg, rrdp, rrdpg`)
- Agent movement (`wmove, smove, wclone, sclone`)

Radio communication is the common trait of the two groups. In particular, instruction in the first group can exchange a number of messages which depends on the number of neighbor nodes before their conclusion (e.g., `routg`). Instead, the latency of agent-movement instructions depends on the size of the agent code

and data structures, which is translated into a variable number of messages to be sent, before the destination node could parse, rebuild the agent and send an acknowledgement which allow the sender to finish the instruction execution.

Those results confirm that, in order to have a fast and lightweight MAMW-based application, it is necessary to optimize those agents which are more prone to be cloned/moved and to avoid remote tuplespace group instructions.

15.2 Agilla Energy-awareness

The energy available to WSN nodes usually comes from batteries with limited capacity (e.g., 1000-3000 mAh). The main source of energy consumption is the the radio transceiver; as it is possible to observe in Figure 15.9, the energy consumption when the radio transceiver is active waiting for incoming transmissions (RX) is about $10 - 15$ times higher than the consumption measured with the transceiver powered off. The consumption raises even further when the transceiver switches to transmission mode (TX).

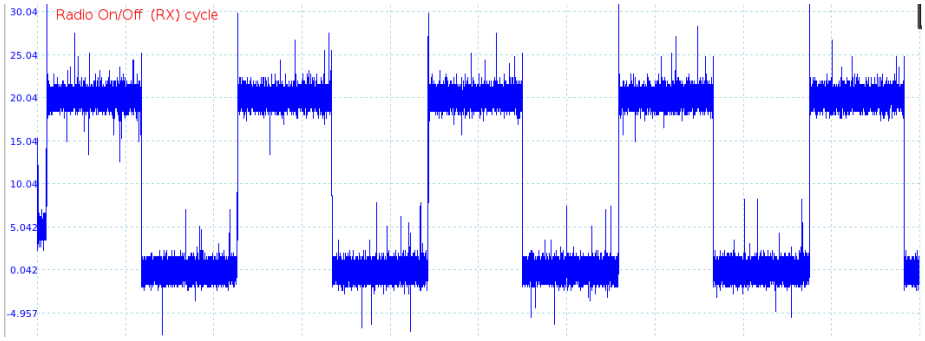


Figure 15.9: WSN node energy consumption (radio ON/OFF cycles)

MAMWs make intense use of the radio transceiver, since both data and agents are transmitted and received through the radio channel. A non-optimized approach in design a MAMW-based application can lead to a (relatively) fast battery wear, at the point that a battery-substitution action is needed.

In order to overcome these issues, this thesis focuses on the following objectives:

- Provide Agilla2 with *Energy-awareness* i.e., the ability to infer the current available energy by reading the state of the batteries
- Design a set of *reconfiguration plans* which can be activated when the battery level is low.

The Agilla *Energy-awareness* has been obtained through the addition of the `battery` instruction in the Agilla ISA. The `battery` instruction reads, through the WSN node ADC, the batteries voltage. Despite the voltage measurement

is not directly useful for measuring the consume of energy, by considering the common discharge curves for batteries, the battery voltage can be used to infer the *state* of the battery itself. In Figure 15.10, four zones can be delimited:

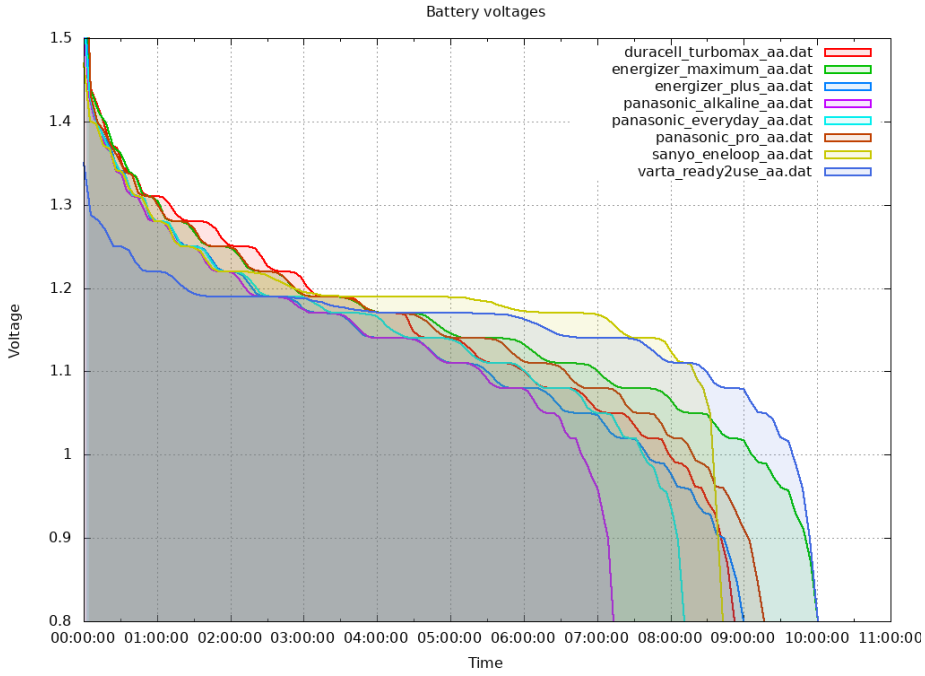


Figure 15.10: Batteries discharge curves

- A *safe* zone, where the battery can be considered charged (1.5V-1.2V)
- A *constant* zone, where the battery voltage is almost constant (1.2V-1.1V)
- A *knee*, where there is a rapid drop in battery voltage
- A *dead* zone, where the current voltage level is not enough to allow digital circuits to work correctly (<1.1V)

Using the aforementioned considerations, the `battery` instruction uses a *threshold* to determine two states: *battery OK* and *Low Battery*. The threshold voltage is set so that there is enough time for the WSN operators to act and change the batteries, e.g., in the *constant* zone of the discharge curves.

In addition to energy-awareness, in order to increase the battery duration the *Proteus Engine*[9] is adopted. *Proteus* is a reconfiguration framework that exploits its language, grounded on a proprietary XSD, aimed at building and managing rules for software reconfiguration. It deals with obtaining, parsing,

interpreting and executing a reconfiguration plans. Every reconfiguration contains information about the properties that nodes under reconfiguration must respect, the reconfiguration action to be taken, and the agent snippet code to be injected.

The scenario is shown in Figure 15.11 (left). A monitoring application, developed as a MAMW application, reads temperature samples from an area of interest. Using Agilla2 and the Proteus Engine, a set of rules (Figure 15.11, on

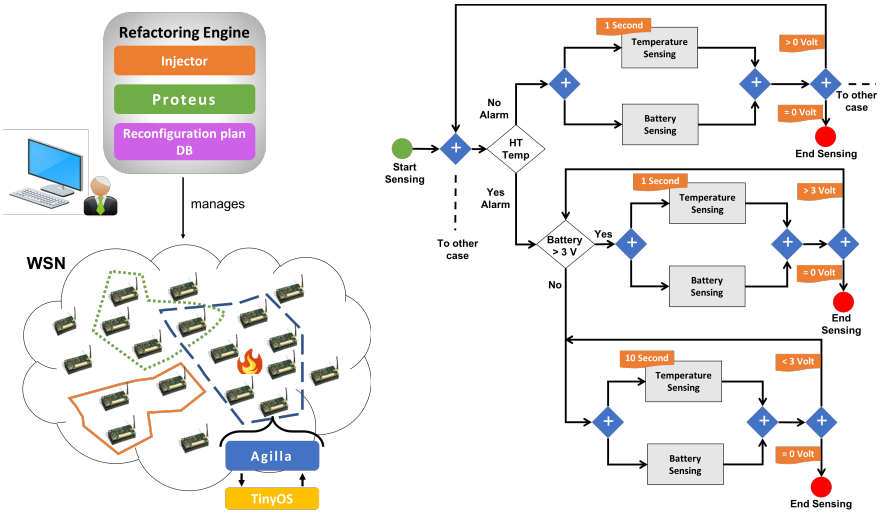


Figure 15.11: Proteus Engine and Agilla: architecture and decision process for reconfiguration

the right) has been defined to decide whether a reconfiguration of the application agents is needed to preserve as much energy as possible. In particular, when this happen, the duty cycle of the temperature readings is decreased to reduce the energy consumption.

The power and energy consumption have been evaluated in different scenarios. The results are shown in Figure 15.12, where the effects of reconfiguration plans on energy consumption can be observed: the measure instantaneous electric current is shown in blue, the energy consumption (in green) is obtained by integrating the current value and by normalizing the result by the voltage value. Figure 15.12 shows the energy consumption in the generic oscilloscope units U and the resulting values converted in one of the commonly adopted energy measurement units, mAh.

15.3 WIDzilla

Using an MAMW for security purposes, as described in Section 13.5, can open the way to multiple techniques useful to dynamically protect the WSN from

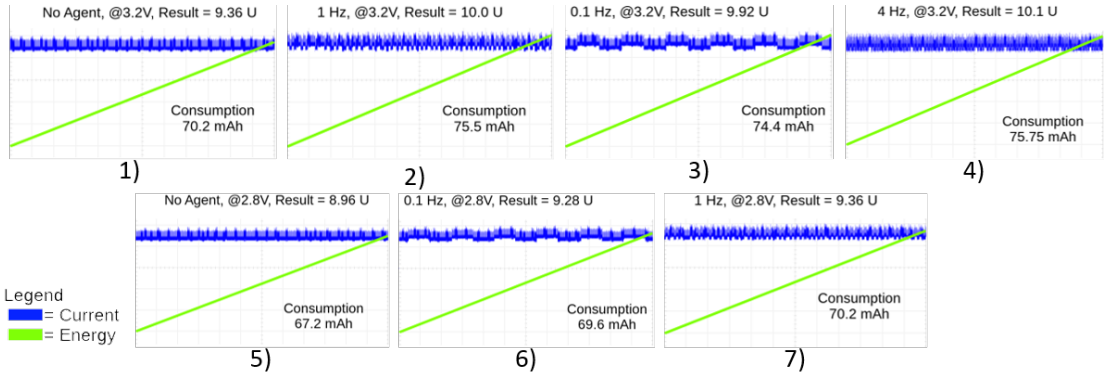


Figure 15.12: Power and energy consumption results

attackers by injecting code (by mean of agents) to quickly react to the attacker moves.

This section describes the research activities related to the incremental design of WIDzilla, a MAMW based on the improved version of Agilla (Agilla2) and the TinyWIDS intrusion detection system, starting from the basic concept and using the tools and techniques in the proposed security framework to produce a full-featured security platform which aims to update the previous works described in [11] and in [12]. Notice that, as it will be pointed out in Section 15.3.2, although various versions of WIDzilla have been designed, at the time of writing, not all those versions are implementable in real WSN nodes.

15.3.1 WIDzilla: incremental design

The first instance of WIDzilla is shown in Figure 15.13. In this section, the TinyWIDS components have been added to Agilla2 and connected TinyWIDS' attack notifications to an *Agilla reaction* which inserts an *Attack Tuple* in the Agilla2 tuplespace of the current node. These tuples can be monitored (locally or remotely by other nodes) and read by agents, which can then perform defence-related actions. For sake of clarity, the TinyOS radio transceiver drivers are represented as a separated component.

Starting from the first WIDzilla concept, the MAMW has been extended to embed also the *passive* security features and the other techniques described in this work to finally obtain the proposed *security framework*. A first improvement has been to adopt a *real* IEEE 802.15.4 MAC layer. The TKN154 has been chosen since that, at the time of writing, it is the MAC layer with the largest number of supported WSN node platforms.

Figure 15.14 shows the resulting software stack. The TKN154 adoption implies Agilla2 to be modified to conform the standard IEEE 802.15.4 interfaces instead of the TinyOS *Active Message*-based primitives.

With the inclusion of a proper MAC layer, the research activities moved

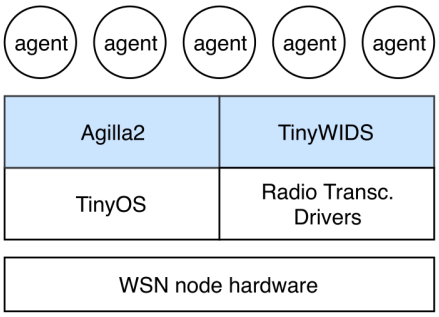


Figure 15.13: (1) First instance of WIDzilla

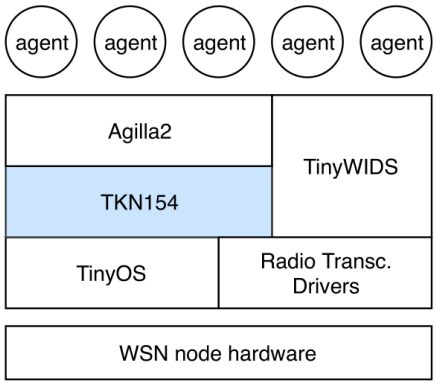


Figure 15.14: (2) WIDzilla with TKN154

forward to design a third version of WIDzilla which also includes the passive security mechanisms, i.e., cryptography. Although the last version of TAKS (Section 10.5.4) could be added on top of TinyOS/TKN154, thanks to research activities described in Section 10.6, it has been decided to include TAKS in the form of a IEEE 802.15.9-compliant KMP and, as requirement, the IEEE 802.15.9 MPX layer. The third version of WIDzilla featuring the IEEE 802.15.9 layer and the TAKS KMP is shown in Figure 15.15.

A variant of this third version (Figure 15.16) replaces the TAKS KMP with the *ECTAKS* KMP, introducing ECC-powered passive security features. Given the IEEE 802.15.9 interfaces, the two KMP could be interchanged easily or having both included and selectable at runtime. In the next paragraphs, however, ECTAKS KMP is assumed to be the adopted KMP.

With the introduction of ECC via ECTAKS and the research activities described in Chapter 12, the next step has been to modify the ECTAKS KMP to rely on the ECC-HAxES hardware accelerator (if present) to perform ECC operations, as depicted in Figure 15.17.

Finally, Figure 15.18 presents the final version of WIDzilla, in which the

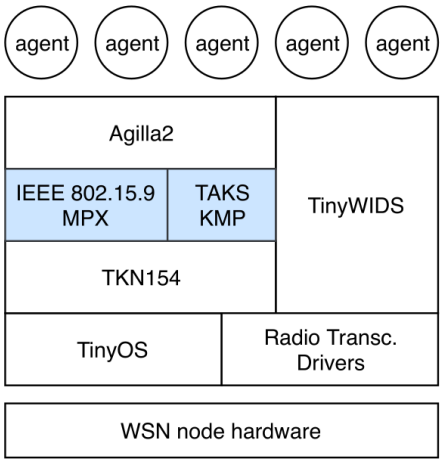


Figure 15.15: (3) Passive security using TAKS KMP via the IEEE 802.15.9 layer

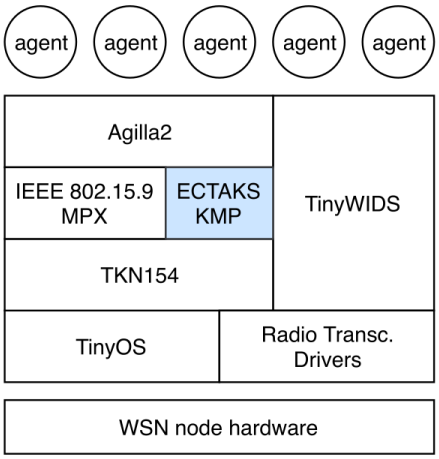


Figure 15.16: (4) Passive security using ECTAKS KMP via the IEEE 802.15.9 layer

Lightweight Blockchain Anti-Tamper mechanism has been included in the MAC layer to protect and filter out messages (and agents) coming from unreliable (and un-trustable) source nodes.

15.3.2 Implementation issues and Future Works

Considering the common WSN node platforms with the biggest amount of Flash storage (128 KB), only the first version of WIDzilla (Figure 15.13) can be implemented with minor problems. The others, instead, presents different issues:

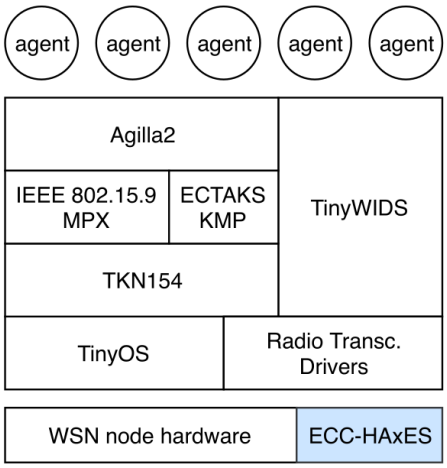


Figure 15.17: (5) Introducing the ECC-HAxES ECC hardware accelerator

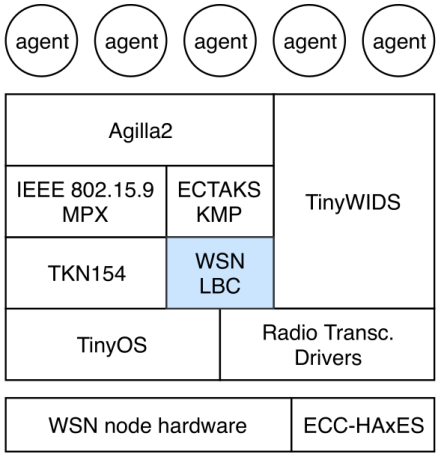


Figure 15.18: (6) Final WIDzilla platform

- The TKN154 platform compatibility (for example, the Iris platform is not supported)
- The amount of memory for code and constant data required is beyond the available capacity.
- Some components may have unsustainable high RAM requirements (e.g., maximum running agents and routing tables for Agilla2 or the WSN-LBC blockchains)

The first issue could be solved by adding support for the required platforms. In particular, some attempts have been made for the Iris platform [102].

Concerning the memory storage requirement, one solution would be to use the larger external Flash/ROM memory, which can contain all the required data, at the price of a slower access (i.e., every chunk of data stored would need to be read and swapped into RAM before use). The *telosb* platform, for example, is equipped with a 1024 KB external serial Flash storage, which would be more than enough for storing data (and agents code). However, using this storage solution for code is not as easy, since in most WSN platforms the MCU can neither execute code directly from the external memory nor re-write the entire code sections at runtime[2].

Future works will analyze new WSN node platforms with increased memory storage to develop the full WIDzilla platform and, eventually, add new features and functionalities to make it the reference security-oriented MAMW.

15.4 Related Publications

The Model-based Porting technique is described in [7], while the detailed description of the Agilla porting process can be found in [94]. The use of Agilla2/3 is proposed in [8] and [9]

[2]Some platforms allows to re-write code sections with some limitations

Part III

Use cases

Chapter 16

VISION

This chapter describes the how the techniques and tools described in the previous chapters found application in the context of the *VISION* project.

16.1 VISION

The *Video-oriented UWB-based Intelligent Ubiquitous Sensing* (VISION) Project [103] is an European Project which aimed to provide real-time sensing services, in particular 3D video sensing, using mobile and context-aware operations. In VISION, the 60 GHz *Ultra-Wide Band* (UWB) radio links are used for broadband communications and for real-time 3D video streaming. The *Ubiquitous Sensing*, apart from video streaming, is provided by a energy-optimized WSN in which WSN nodes perform light, temperature, audio and other sensor readings. Due to the UWB issues, in VISION, a set of cross-layer optimizations have been provided to achieve the best possible *Quality of Service* (QoS). The VISION WSN, empowered by real-time video streaming through UWB radio channels gave the birth to the so-called *Wireless Multimedia Sensor Network* (WMSN).

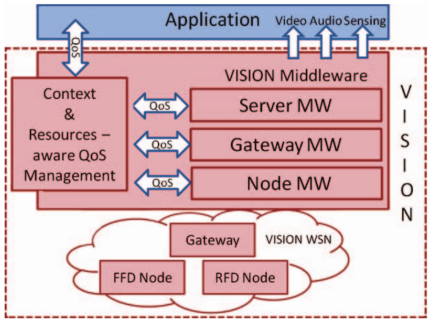


Figure 16.1: VISION platform overview

The necessity of such a level of optimization is satisfied in VISION by the *WSN Manager* which employs a MW to apply dynamic *reconfiguration plans* to the WSN. Those plans are meant to dynamically adjust the QoS and/or the energy consumption.

An overview of the VISION project platform is shown in Figure 16.1.

16.2 Agilla MW in VISION

The part of the VISION MW running on WSN nodes was provided by the Agilla2 MW, which, thanks to the research activities described in Section 15.2, is capable of running the required configuration plans and monitoring the current power supply (i.e., batteries) status.

During the research activities in the context of the VISION project, a new feature has been introduced in Agilla2, the *Morse-based* communication via the LEDs embedded in the WSN nodes. A new agent instruction (`morse`) has been added to Agilla2 ISA to translate a short ASCII message (which is pushed in the Agilla2 operand stack) into a Morse-encoded sequence of LEDs blinks. This feature is used in the VISION use-case (described in the next section) to provide an uni-directional communication channel which is used by the WSN nodes to signal their presence and identity when harsh conditions (e.g., smoke presence) are detected. The `morse` instruction is issued by a proper agent which gets injected as consequence of an emergency reconfiguration-plan.

16.3 VISION *Fire Rescue* Use case

The main VISION use case is the *Fire rescue* scenario, shown in Figure 16.2. In a building which is supposed to be on fire, a remote-controlled *Robot* is released to help rescuers to locate and save endangered people currently in the building. The robot is equipped with a UWB radio transceiver, a set of cameras (one of which is an *infrared camera* providing real-time video even with the presence of smoke/fire) and a WSN sink node. The Robot act as a FFD and as a bridge between IEEE 802.15.4 WSN and the UWB receiver in the main operation server. The camera is also used as mean to detect WSN node presence and identity by decoding their morse-based communication. The information coming from the robot, from the cameras and from the WSN are combined together to provide rescues with an *augmented reality* video which can be visualized by rescuers both from the main server or via virtual reality glasses. This augmented video can be used to find e.g., the shortest or safest path to reach endangered people.

In the use-case, the dynamic reconfigurations happen in the following scenarios:

- When no fire is detected, the WSN nodes sends periodically sensor data to the sink node. When a low battery status is detected, a reconfiguration

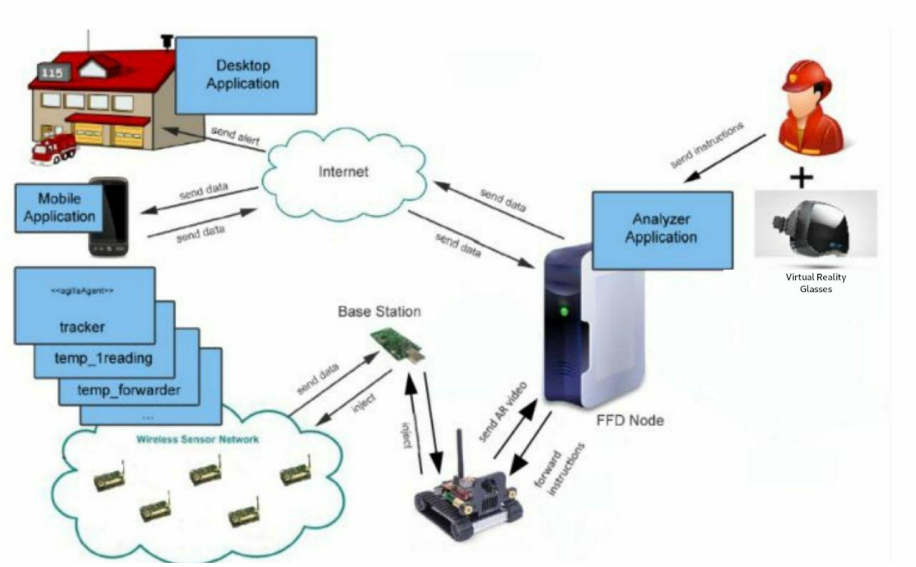


Figure 16.2: VISION: *Fire-rescue* scenario

plan is adopted to modify the sampling frequency, reducing the energy consumption.

- When fire is detected (e.g., by room smoke sensors), a reconfiguration plan is adopted to flood the WSN with *fire tracking* agents, which are used to gather information on the affected area.
- Also, a reconfiguration is performed when additional energy-hungry sensors are needed to facilitate the rescuing operations.
- Finally, a reconfiguration plan can be adopted to inject the *morse agent* in the WSN.

16.4 Results

The VISION infrastructure has been successfully demonstrated as final part of the project. The enhanced Agilla2 (with the new `morse` instruction) has been validated and actively used in the context of the demonstration on Iris WSN motes, equipped with a MDA100CB sensorboard modified to embed a bigger red LEDs used to blink the morse-encoded messages on request (Figure 16.3).

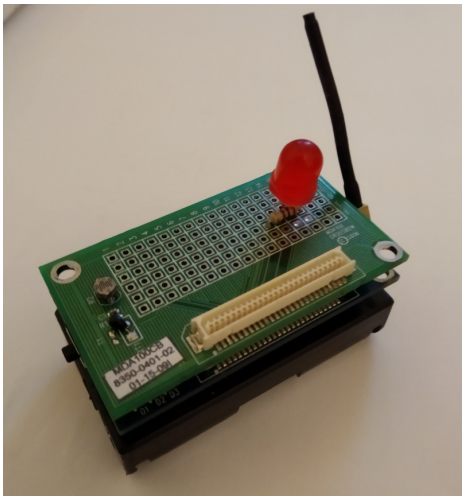


Figure 16.3: VISION: modified MDA100CB sensorboard on an Iris node

Chapter 17

SEAMLESS

This chapter describes the SEAMLESS project and how the technologies described in the previous chapters took part in it.

17.1 SEAMLESS

SEAMLESS is a National Italian project in the context of the Military National Research Program. SEAMLESS objective was to develop a *secure platform* formed by a WSN platform, a management system and a set of monitoring tools. SEAMLESS was carried out by the Italian company *RoTechnology* with the support of the University of L'Aquila as technology consultant.

An overview on SEAMLESS is shown in Figure 17.1.

SEAMLESS WSN uses TinyOS and the TKN154 MAC layer [31] to realize a *Cluster-Tree* or, optionally, a *Mesh* topology. To do so, SEAMLESS introduces in TinyOS the support for the previously unsupported *Iris* platform for the TKN154, an implementation of the *Ad-Hoc On-Demand Distance Vector* (AODV) routing protocol and the *Multi-Hop* communications support for mesh topologies.

Each node in the SEAMLESS WSN mounts the *MTS420CC*, a sensorboard equipped with various sensors, including a temperature/humidity sensor, a GPS and a 3-axis accelerometers. These sensors are read by the node and sent to a message queue and a message broker (using the MQTT protocol [111]) to a server which offers network operators an HTTP monitoring interface (i.e., a *Dashboard*) which can be consulted to visualize sensor data on different formats, including a geographical representation. An example is shown in Figure 17.2.

From the security point of view, TAKS is adopted to provide both lightweight encryption and authentication. The inner symmetric algorithm used in TAKS is AES 128bit in CTR mode and *HMAC* MAC function is used to generate the authentication tags. As active security measure, instead, WIDS is used to detect *jamming*, *Sybil* or general *Spoofing* attacks.

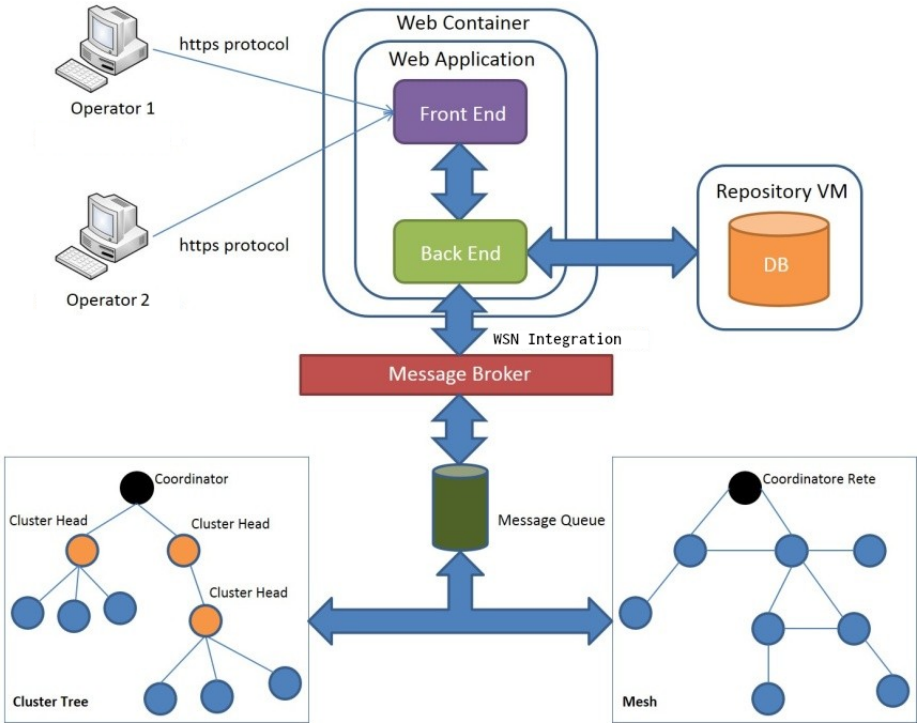


Figure 17.1: SEAMLESS overview

17.2 Results

The SEAMLESS Project has been successful: the WSN confidentiality (provided by TAKS) was tested against sniffing-based attacks (i.e., using nodes programmed to catch every valid frame) while the intrusion detection was tested by simulating jamming attacks, replay attacks, sybil attacks and spoofing. When one of such attacks has been detected, attack information and the nodes under attacks are displayed in the Dashboard to alert the network operators.

17.3 Related publications

Additional information on SEAMLESS can be found in the paper presented by authors in the ITASEC 2020 Italian Cyber-Security Conference [5].

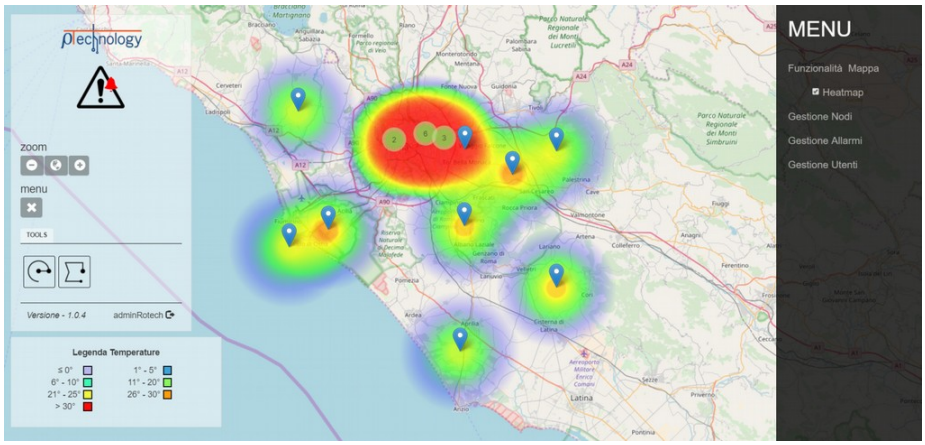


Figure 17.2: SEAMLESS: geographical visualization of sensor data

Chapter 18

SafeCOP

The *Safe Cooperating Cyber-physical Systems using Wireless Communication* (**SafeCOP**[115]) is an ECSEL European Project in which the *cooperation* of Cyber-physical systems (CPSs) is adopted as mean to provide security and safety assurance. The cooperating CPSs are referred as *CO-CPSs*. To do so, SafeCOP provides *methods* and *tools*, and extensions to technologies towards safety of CPSs. In general, SafeCOP's objective have been:

- propose a *safety-assurance framework* to facilitate the safety certification process;
- develop a *Runtime Manager* used to assert the normal behavior of the target CO-CPS, triggering and switch the CO-CPSs to a safer *degraded mode* when needed, i.e. a operational mode which trades performances or efficiency for safety;
- extends the wireless communication protocols towards *security* and safety;
- provide contributions to current standards and regulations;
- provide a set of *real world* scenarios to demonstrate the SafeCOP approach.

A complete overview on SafeCOP is shown in Figure 18.1. SafeCOP started in April 2016 and ended in June 2019 involving 28 partners from 6 different European countries.

18.1 SafeCOP Use Case 5

The SafeCOP UC5 (*V2I Cooperation for Traffic Management*) demonstrates the effectiveness of the SafeCOP approach in Vehicle-to-Infrastructure communication for traffic management. In this context, different *Cooperative Intelligent Transport Systems* (C-ITS) communicates together to exchange information and to warn about possible road problems. This increases the traffic management

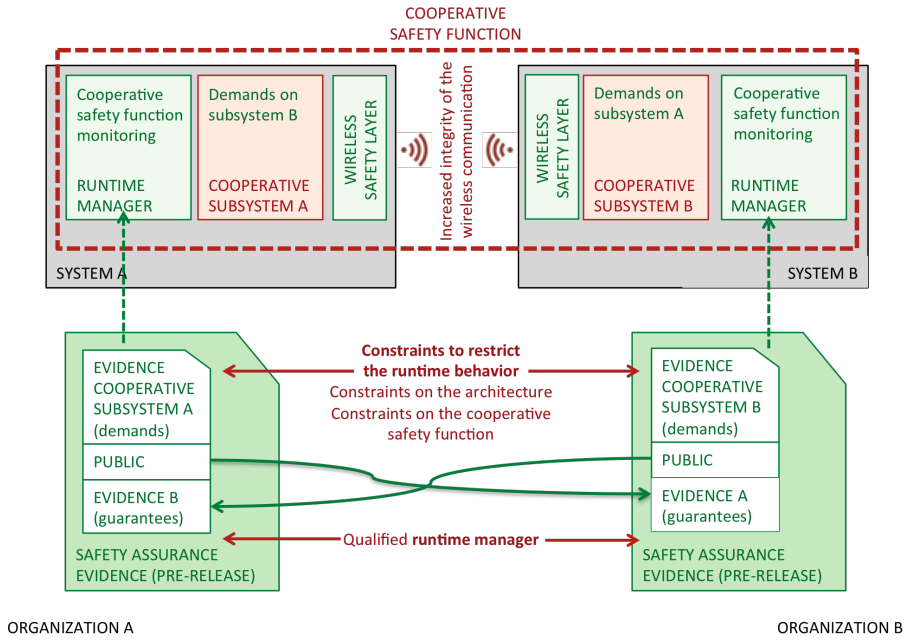


Figure 18.1: SafeCOP: overview

effectiveness but, in general, increases also the safety concerns. On this side, the SafeCOP approach tries to:

- adopt new technologies to improve security and safety assurance;
- standardize methods and techniques to favor utilization.

In the UC5, one of the key techniques is the *sensor fusion*: different technologies *cooperate* together to provide the *Runtime Manager* with data useful to decide a reaction, eventually causing the C-ITS to switch to the *degraded mode* to ensure the safest possible behavior for the C-ITS.

The UC5 scenario is shown in Figure 18.2.

The technologies adopted in the UC5 are the following:

- a *Local Control Unit* (LCU) which collects data and decides (using the *Runtime Manager*) which kind of actions to perform.
- *Video Content Analysis* (VCA) on the roadside through the use of cameras.
- *On-Board Units* (OBU) installed on the vehicles. The OBUs provide individual vehicle data, such as *accelerations*, *speeds*, *position* etc. These data is *fused* to produce derived measures of the vehicle behavior.

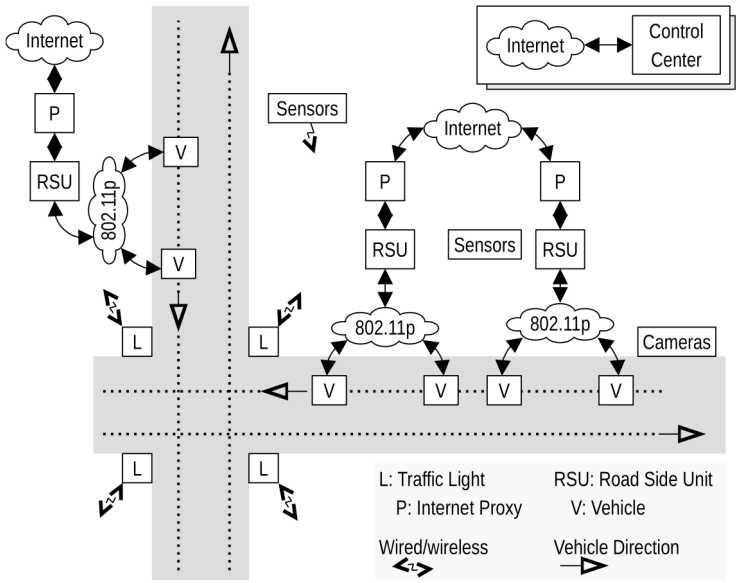


Figure 18.2: SafeCOP UC5: overview

- *Adaptive Traffic Light* (A-TLS) and the *Green Light Optimal Speed Advisory* (GLOSA) to actively control the traffic flow.
- Roadside *Sensor Networks* (RSU-SN) to monitor the environmental parameters in the areas in proximity of the road.

This last technology has been deployed through the use of WSN and it will described in details in the next section.

18.1.1 UC5 Road-side Unit - Sensor Network (RSU-SN)

The RSU-SN is realized by a star-topology WSN deployed in the area around the road/crossroad of interest. Each WSN mote is equipped with five different sensors:

- Light sensor (Lumen)
- Temperature sensor (Celsius degrees)
- Humidity sensor (percentage)
- Fog presence sensor (percentage)
- Ice presence sensor (percentage)

The light sensor is used to detect the amount of the current environmental light, which is useful to detect an eventual dark road situation that could represent a

safety issue. The temperature and the humidity sensor are used to detect safety issues that could arise when it is raining. The fog sensor is a *virtual* sensor, which is created by combining the temperature, the humidity and the seasonal dew point. Finally, the ice sensor is used to retrieve a percentage of ice presence on the road, which is a critical information when safety is concerned.

The WSN is show in Figure 18.3. The sink node (center of the star) gathers data from the device nodes and forwards them to a *single-board-computer* (SBC) connected via serial port. The SBC perform basic tests on data and filters erroneous or corrupted data out before sending them to the LCU.

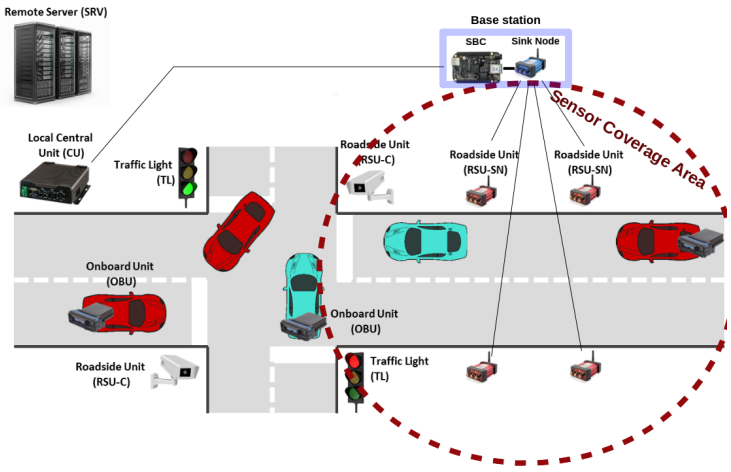


Figure 18.3: SafeCOP UC5: RSU-SN

The WSN node platforms selected has been the *Memsic Iris* nodes [99]. Every communication among WSN nodes is based on the IEEE 802.15.4 standard and it is encrypted through TAKS and monitored though TinyWIDS to detect intruders.

Communications between the SBC and the LCU are instead based on conventional networking protocols (e.g., Ethernet, WIFI or IEEE 802.11p), using the *Message Queue Telemetry Transport* (MQTT [111]) protocol, using a specific syntax defined within the UC5 scenario.

18.2 Results

The SafeCOP UC5 has been part of the SafeCOP final demonstration. The demonstration has successfully shown the cooperation among the UC5 technologies, the ability to communicate with the LCU and the switch of every component to the *degraded mode* when the safety-concerning scenarios has been simulated. TAKS and TinyWIDS have been validated successfully during the

demonstration by simulating sniffing attacks, jamming attacks and replay attacks. Sniffing attacks failed (i.e., the attacker could not retrieve the plaintext messages) thanks to TAKS, while TinyWIDS successfully detected the other attacks, sending notifications causing the LCU to switch the RSU-SN to degraded mode.

Chapter 19

DESTAK

This chapter presents the *TAKS over DSME* (DESTAK), a project aimed to provide a secure environment over the IEEE 802.15.4 MAC Behavior DSME using hybrid cryptography schemes. After a brief description of the idea, we present the design and the integration of TAKS in the DSME, its implementation using a network simulator and some exprerimental results.

19.1 DESTAK: TAKS over DSME

DESTAK is a project resulting from a collaboration between the DEWS center of the University of L'Aquila and the CISTER of the ISEP/IPP of Porto. DESTAK aims to provide a lightweight security solution (with encryption, authentication and key management) for *Industrial IoT* protocols. In particular, DESTAK focuses on the IEEE 802.15.4 *Deterministic Synchronous Multichannel Extension* (DSME) MAC behavior [16]. The DSME add *Multi-channel* access to the radio medium via frequency hopping and channel adaptation (i.e., dynamic allocation of channels depending on the channel states). Also, DSME adds other features e.g., *Multi-superframes*, *Group Acknowledgements*, *Enhanced Beacons* to delimit multi-superframes, *CAP reduction*, etc.

DSME idea can be observed in Figure 19.1, where two *Multi-superframe* are shown. In green, the GTSs reserved to a single node are shown.

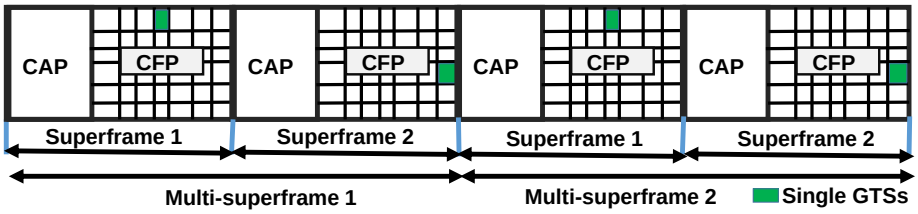


Figure 19.1: DSME multi superframe structure

19.2 OpenDSME

OpenDSME [92] is a DSME simulator based on the famous *Omnet++* project [93]. In OpenDSME, there are several compound modules such as the DSME data link layer which handles the GTSs allocations. One of the major limitations of OpenDSME is the lack of security modules.

19.3 *Secure* OpenDSME

In order to provide a lightweight security mechanism to DSME, TAKS has been adopted to be implemented in the OpenDSME simulator.

The adaptation and implementation have been performed as follows.

1. A set of classes for providing a C++ version of TAKS have been designed and implemented
2. A new set of *Information Elements* (IEs) have been designed to fit the limitations and the requirements of TAKS and DSME. As a result, a new MAC frame format which includes the TAKS IEs has been determined and proper classes created (Figure 19.5).
3. Additional classes have been created to model application/higher level protocol payloads.
4. The TAKS classes have been integrated into the OpenDSME code to encrypt/tag outgoing data frames and decrypting/authenticate incoming frames using TAKS and the TAKS IEs.

19.3.1 TAKS for Omnet++

TAKS has been implemented in C++ to be inserted and compiled with the Omnet++ framework. In particular, TAKS key components have been modeled though the introduction of the `TAKSComponent<nbits>` class. This class contains all the methods required to store and use a key component of arbitrary bit length. The TAKS encryption and decryption function are instead implemented through the singleton class `Taks<>`. This class also contains the TAK function, the symmetric encryption/decryption and the authentication functions.

19.3.2 TAKS Information Elements

IEEE 802.15.4 IEs in a nutshell

In the IEEE 802.15.4 standard, the IEs are optional fields which can appear both in the MAC header or at the beginning of the payload. In the first case, they are called *Header IEs*, while in the latter they are called *Payload IEs*.

Despite the differences, an IE usually contains upper-layer information and data which cannot be entirely considered a *payload*. The presence of one or

more IEs is signaled by the `IE present` field in the Frame Control Field. If `IE present=1`, after the conventional header fields, *IE headers* (eventually followed by the IE contents) are found. Also, special headers (called *IE Terminations*) are appended at the end of the IEs to signal the beginning of the real MAC payload.

An *Header IE* starts with a 2-byte header composed by:

- a `length` field, which contains the length (in bytes) of the IE
- a `elementID` field, which denote the specific IE. Some pre-defined IDs are listed in the standard
- a `type` field, which is a single bit to specify if the IE header refers to a Header IE (`type=0`) or a Payload IE (`type=1`)

After the header, the content of the IE can be appended. At the end of all the Header IEs, the 2-byte Header Termination *HT2* is required.

Payload IEs are slightly more complex. First of all, in the MAC header the Header Termination *HT1* needs to be inserted. Then, preceding the former MAC payload, the *Payload IE Headers* and the relative contents can be specified. The Payload IE Header structure is similar to the Header IEs (only the bit-length of the fields changes), with only the `type` field content as real difference. After all the Payload IEs, a special termination header (called *Payload Termination* needs to be appended before the MAC payload.

TAKS IEs definition

In order to insert the information required by TAKS in addition to the ciphertext (i.e., the *Key Reconstruction Information* KRI and the Authentication Tag τ), the following approaches have been analyzed:

- Appending KRI and τ in the MAC payload
- Using Header IEs for KRI and τ, placing the ciphertext in the MAC payload
- Using Payload IEs for KRI and τ, then appending the ciphertext in the MAC payload

The first option have been discarded, since it is a un-structured approach that could bring to frame parsing problems. The other two options are instead shown in Figure 19.2 and in Figure 19.3.

Figure 19.2: TAKS IEs: Header IE case

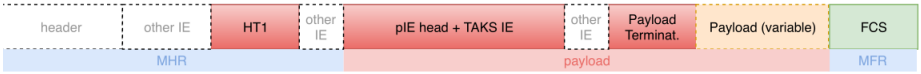


Figure 19.3: TAKS IEs: Payload IE case

Due the very limited space available in the MAC frame, the Header IE option has been selected, since it saves few bytes (a small but important amount in respect of the available space). Fixing the TAKS key length to 128 bits and using $d = 2$, the total amount of space for ciphertext can be estimated to be ~ 70 bytes or ~ 50 bytes if extended addresses are used. The final IEEE 802.15.4 frame structure is shown in Figure 19.4

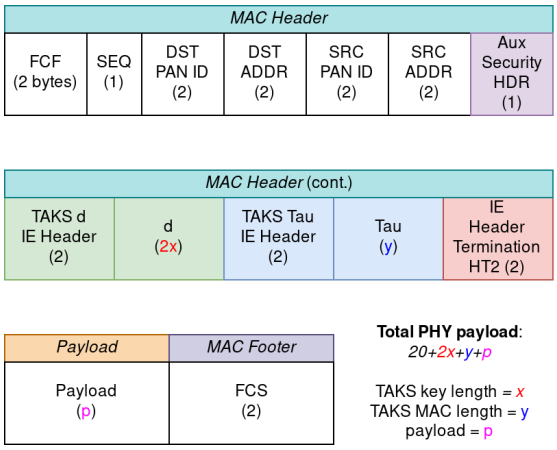


Figure 19.4: DESTAK IEEE 802.15.4 frame

Finally,the required C++ classes for modeling IEs, Header IEs, Payload IEs and the specific TAKS IEs have been defined and implemented. Proper methods for parsing and serializing those classes have also been developed for later inclusion in OpenDSME.

19.3.3 Payload representation

Since the payload is left un-modeled by OpenDSME, simple storage class has been designed to uniform the overall Secure DSME platform.

19.3.4 Integration in OpenDSME

The integration of TAKS into OpenDSME has been performed in three steps:

1. All the class described in the previous sections have been added in OpenDSME. The resulting integration is shown in Figure 19.5.

2. The finite state machine which handles outgoing *data-only* frames has been injected with the code required to perform TAKS encryption and to append the resulting TAKS IEs.

3. The finite state machine which handles incoming frames has been injected with the IE parsing procedures and the TAKS decryption.

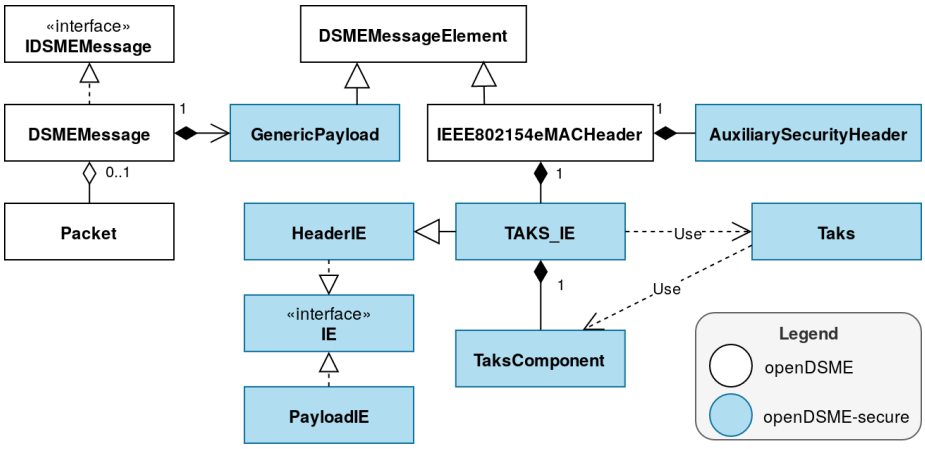


Figure 19.5: TAKS IEs: Payload IE case

19.4 Results

In order to validate the simulator, the correct encryption/decryption of data frames and to evaluate the performance impact of TAKS in OpenDSME, a configurable Omnet++ simulation has been created. In this simulation, a separate script is used to configure different values for the following parameters:

- The TAKS key length (64, 96, 128 bits)
- The number of nodes in the simulated WSN
- The number of frames generated by every node
- The MAC *Beacon Order* (BO), *Superframe Order* (SO) and the DSME-specific *Multi-superframe Order* (MO)
- The simulation total time

Figure 19.6, shows the comparison of throughput done for nodes ranging from 5-30. The setup with no security was able to provide larger throughput comparatively. The experiment done with the highest order of security (128 bit key length) was able to obtain around 10% lesser than the one without the

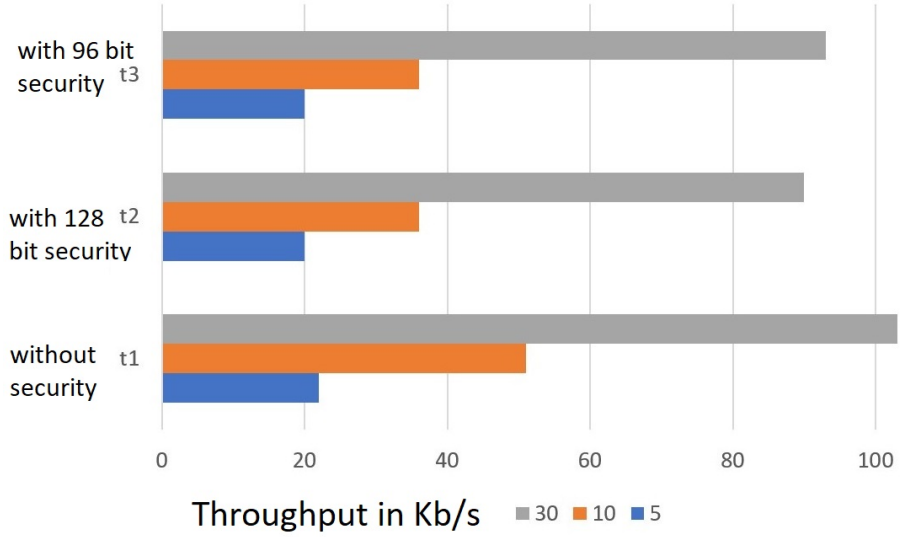


Figure 19.6: DESTAK results

security; in the case of lesser nodes (5 nodes), the reduction of throughput was only around 5%. From this test, it very clear that TAKS is an ideal mechanism to support these low power networks for a smaller number of nodes as it will greatly compromise on throughput.

19.5 Related publications

DESTAK is described in the technical report [6], which is currently a work in progress.

Part IV

Conclusions

Chapter 20

Conclusions

In this PhD thesis, a framework for proving security to WSNs has been proposed. This framework embeds different features, tools, approaches and techniques which can be adopted and combined to fit the requirements of each WSN application. In particular, this thesis presented lightweight cryptographic schemes, a cryptographic hardware accelerator, intrusion detection systems and a anti-tampering mechanism based on blockchains for WSN. In order to increase the flexibility and the cooperation of such components, an agent-based middleware has been adopted and enhanced. The framework has been then adopted to create a WSN secure platform which contains all the above mentioned components. The framework and the components it provides, have been validated through various test cases, including the adoption of them in the context of different European Research Projects (VISION, SEAMLESS, SafeCOP) deliverables and use-cases.

The punctual list of the described contributions follows.

1. *The TAKSv2 re-development, enhancement, on-field validation and performance evaluation.* Starting from [11] and other previous works, TAKS has been re-developed and enhanced to extend its compatibility with software platforms (e.g., TinyOSv2) with a new software architecture that allows to easily integrate future contributions.

2. *A secure random number generator for TAKS.* In [19], despite to the presence of randomly-generated values, no details are present on how they are generated. This thesis proposed two new lightweight CSPRNG to be used in TAKS.

3. *The IEEE 802.15.9-compliant TAKS KMP design.* Considering the release of the IEEE 802.15.9 standard for *Cryptographic Key Transportation*, this thesis proposed a new KMP design featuring TAKS.

4. *The design of OpenZB-enabled TAKS.* This thesis proposed also a adapted version of TAKS which is integrated to the OpenZB software platform to

provide hybrid cryptography encryption, authentication and key management.

5. *The Elliptic Curve TAKS (ECTAKS) development, validation and performance evaluation.* This thesis proposed a new version of TAKS extended to used Elliptic Curve Cryptography as base. This version has been implemented in TinyOS by means of the TinyECC library and validated.

6. *The novel TAKS/ECTAKS key components generator.* In [19], the TAKS key components are generated with an outdated generator. This thesis proposed a new Python-based key components generator which is very fast and able to generate key components for every logical topology chosen.

7. *The development, validation and performance evaluation of a novel hardware accelerator for elliptic curve cryptography for WSN nodes (ECC-HAxES).* Due to the limitation and the performance impact of ECC, this thesis proposed a new approach featuring the design of a ECC hardware accelerator for WSN nodes. This accelerator has a low-area policy and can be deployed in low-power FPGA platforms connected to the WSN node.

8. *The implementation of WIDS on TinyOS (TinyWIDS).* With respect to [13]. This thesis proposed a new TinyOS-based implementation of WIDS with additional features (e.g., Metrics, Rule-generated Observables, etc.).

9. *The definition of a syntax for describing attacks in TinyWIDS.* In [13], no standard methodology to describe attacks as WPM is defined. This thesis proposed a JSON-based syntax in TinyWIDS which is parsed to automatically create a WPM describing the target attack.

10. *The development, validation and performance evaluation of a novel Anti-tampering blockchain-based mechanism for WSNs.* This thesis proposed a novel anti-tampering mechanism which makes use of the blockchain technology to detect bed-behaving WSN nodes and excluding them from the WSN communications. This mechanism is also able to detect injected/captured nodes.

11. *The development, validation and performance evaluation of the TinyOSv2-compatible version of the Agilla MAMW (Agilla2).* The Agilla MW is not natively compatible with the last version of the TinyOS operating system. This thesis proposed a new version of it resulted from the porting of the old Agilla and the addition of new useful features.

12. *The definition of a new model-based methodology to perform the porting of WSN software application (Model-based Porting).* The porting operations described in the previous point led to the definition of a general methodology to support the porting of WSN software application to different hardware or software platforms. This methodology makes use of a model-based technique to locate, inside the target application, the components requiring the porting and guide the porting operations.

13. *The design of a security-oriented MAMW (WIDzilla).* This thesis proposed an integration of the tools and techniques of the proposed framework in a novel security-oriented MAMW for WSN.

14. *The development, validation and performance evaluation of a DSME-compatible version of TAKS in an Omnet++-based simulator (DESTAK).* Considering the industrial applications of WSNs, this thesis proposed the integration of TAKS into the *Deterministic Synchronous Multi-channel Extension* (DSME) IEEE 802.15.4 MAC Behavior. TAKS has been integrated into an Omnet++ based simulator (OpenDSME), validated and evaluated in its performances.

15. *Contributions to the VISION project deliverables and demonstrator.*

16. *Contributions to the SEAMLESS project deliverables.*

17. *Contributions to the SafeCOP project deliverables and to the UC5 use-case and demonstrator.*

20.1 Future Works

Starting from the results of this thesis, below a list of possible future research activities is presented.

- TAKS improvement: *Post-quantum (EC)TAKS*: with the *Quantum Computers* and thanks to the *Shor Algorithm*, it will be possible to reduce the computation time required to perform an exhaustive linear research (*bruteforce*) to solve the factorization and the discrete logarithm problems, causing all the current era public-key cryptography to stop being secure. However, the research world is actively searching for quantum-resistant cryptography, i.e., cryptographic schemes which maintain their security level even when attacked with a quantum computer. In this context, one of the most promising quantum-resistant problem is based on the computation of *elliptic curve isogenies*. A future version of TAKS/ECTAKS could be developed to face the quantum world and, at the same time, keeping a lightweight profile, so that it could be kept to be usable in resource-constrained platforms.

- *Improved WSN nodes with FPGA*. As described in Chapter 12, in this thesis a hardware accelerator on a reconfigurable platform for WSN nodes is proposed. However, in the future, two actions could be taken to improve the cooperation between the WSN nodes and their accelerators: first, the FPGA chip could be mounted directly on the WSN MCU and radio transceiver to allow developers to have an embedded and wireless-interconnected reconfigurable platform which could host accelerators of any kind. A second improvement could be the deployment of the accelerator *and* the MCU/transceiver in a single chip.

- *New MW with High-level language for WSN MW.* One practical issue with Agilla and our enhanced versions is the lack of a proper abstraction level of the programming language used for developing agents. In fact, Agilla Agent Language is an assembly-like language, with no high-level constructs, no software engineering concepts and redundant instructions. A future advancement would be to exploit common high-level (interpreted) programming languages and their bytecode as agent language. For example, it could be possible to exploit e.g., the Python language to write agents, convert the Python code into Python byte code and *code objects* that could be interpreted by the MAMW engine. A challenge, in this case, would be to adapt the interpreter to the resource-constrained WSN nodes.

20.2 List of Pubblications

- Conference Paper: Bozzi, Luciano & Giuseppe, Lorenzo & Pomante, Luigi & Pugliese, Marco & Santic, Marco & Santucci, Fortunato & Tiberti, Walter. (2018). TinyWIDS: a WPM-based Intrusion Detection System for TinyOS2.x/802.15.4 Wireless Sensor Networks. 13-16. 10.1145/3178291.3178293.

- Conference Paper: W. Tiberti, A. Camrmenini, D. Cassioli, A Lightweight Blockchain Technique for Anti-Tampering in Wireless Sensor Networks, submitted to the ITASEC 2020 Italian Conference on Cyber-Security, 2020, [4]

- Conference Paper: M. Pugliese, L. Bozzi, L. Pomante, W. Tiberti, and F. Santucci, SEAMLESS Project: Development of a Performing Secure Platform for IEEE 802.15.4 Network Applications, submitted to the ITASEC 2020 Italian Conference on Cyber-Security, 2020 [5]

- Conference Paper: W. Tiberti, H. Kurunathan. "DeSTAK - A secure approach for low power deterministic networks". Work in progress, [6]

- Conference Paper: Pomante L., Santic M., Tiberti W., "A Renovated Mobile Agents Middleware for WSN - Porting of Agilla to the TinyOS 2.x Platform" in IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI) (IEEE RTSI 2016)

- Journal Paper: W. Tiberti, F. Caruso, L. Pomante, M. Pugliese, M. Santic, F. Santucci, Development of an extended *Topology-based* Lightweight Cryptographic Scheme for IEEE 802.15.4 Wireless Sensor Networks, submitted to the International Journal of Distributed Sensor Networks, 2019 [3]

- Journal Paper: D. Cassioli, A. Di Marco, L. Pomante, M. Santic, W. Tiberti. A *Model-based* porting of WSN middleware: the Agilla experience, Work in progress, [7]

- Journal Paper: Berardinelli L., Di Marco A., Pace S., Pomante L., Tiberti W. "Energy Consumption Analysis and Design of Energy-Aware WSN Agents in fUML", Modelling Foundations and Applications, Springer International Publishing, 2015, 10.1007/978-3-319-21151-0_1

- Journal Paper: Vittorio Cortellessa, Antinisca Di Marco, Daniele Di Pompeo, Francesco Gallo, Stefano Pace, Luigi Pomante, and Walter Tiberti. 2018. Energy-Driven Reconfiguration of Applications for Wireless Sensor Networks. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18). ACM, New York, NY, USA, 79-84. DOI: https://doi.org/10.1145/3185768.3186312

- Journal Paper: Barile G., Leoni A., Muttillo M., Sulli V., Tiberti W. : "SPOF—Slave Powerlink on FPGA for Smart Sensors and Actuators Interfacing for Industry 4.0", Special Issue "Energy Efficient Systems, Sensors, and Smart Management Approaches for Industry 4.0, (work done in collaboration with a different research group)

- Journal Paper: (multiple authors), "Safety and Security in V2I Traffic Management: The SafeCOP Approach", Work in progress, [122]

- Poster: TinyWIDS - DATE 2019 University Booth, *http://www.safecop.eu/wp-content/uploads/2019/03/conference_poster_UNIVAQ.pdf*

- Poster: "A Mobile-Agent Security-Oriented Middleware for WSN", ACACES 2017 (presentation was done by the student)

- Poster: "A Real-Time and Mixed Criticality Extension for a System-Level HW/SW Co-Design Methodology", ACACES 2017

- Poster: "A Secure Mobile-Agent Middleware platform for WSN" (HiPEAC 2018, presentation was done by the student)

- Poster: Santic M., Pomante L., Tiberti W., Centofanti C.: "Labsmiling: a framework, composed of a remotely accessible testbed and related sw tools, for analysis and design of low data-rate wireless personal area networks based on ieee 802.15.4" in Design, Automation and test in Europe conference (DATE), Lausanne (CH) 27-31 March 2017

Acknowledgements

The author would like to thank all the people, colleagues, researchers, professors and technical staff which helped in some way to any of the research activities presented in this work. In particular, I would like to thank

my advisor Luigi Pomante for his guidance, support and advices through my Laurea, Laurea Magistrale and this PhD; my co-advisors Antinisca Di Marco and Marco Santic, Prof. Fortunato Santucci and Dr. Marco Pugliese for the help, information, advices, encouragement and support during all the research activities. We would like also to thank Dr. Stefano Marchesani, Alessandro Ranalli, Eugenio Carocci, Lorenzo Di Giuseppe, Luciano Bozzi and Fabio Del Forno for their previous works on TAKS, WIDS and in the context of the SEAMLESS project; Dr. Dajana Cassioli for her support and collaboration in various activities; Proff. Norberto Gavioli and Riccardo Aragona for their insights and hints on TAKS theoretical security; Dr. Ricardo Severino, Harrison Kurunathan and all the great CISTER researchers and staff for all the support given during author's exchange period; Dr. Carlo Brandolese and all the people involved in the SafeCOP UC5.

Last but not least, I would like to thank my beloved Roberta, who, with her support, encouragement, patience and understanding, enlighted me during the writing part of this thesis.

Bibliography

[1] Bozzi, Luciano & Giuseppe, Lorenzo & Pomante, Luigi & Pugliese, Marco & Santic, Marco & Santucci, Fortunato & Tiberti, Walter. (2018). TinyWIDS: a WPM-based Intrusion Detection System for TinyOS2.x/802.15.4 Wireless Sensor Networks. 13-16. 10.1145/3178291.3178293.

[2] TinyWIDS - DATE 2019 University Booth, http://www.safecop.eu/wp-content/uploads/2019/03/conference_poster_UNIVAQ.pdf

[3] W. Tiberti, F. Caruso, L. Pomante, M. Pugliese, M. Santic, F. Santucci, Development of an extended *Topology-based* Lightweight Cryptographic Scheme for IEEE 802.15.4 Wireless Sensor Networks, submitted to the International Journal of Distributed Sensor Networks, 2019 https://univaq-my.sharepoint.com/:f:/g/personal/walter_tiberti_univaq_it/EihenFEx2yNDp4BcSoVRNVkBxvtlLhBhe9srfzwJbrAv3w?e=zYZ20o

[4] W. Tiberti, A. Camrmenini, D. Cassioli, A Lightweight Blockchain Technique for Anti-Tampering in Wireless Sensor Networks, submitted to the ITASEC 2020 Italian Conference on Cyber-Security, 2020, https://univaq-my.sharepoint.com/:f:/g/personal/walter_tiberti_univaq_it/EihenFEx2yNDp4BcSoVRNVkBxvtlLhBhe9srfzwJbrAv3w?e=zYZ20o

[5] M. Pugliese, L. Bozzi, L. Pomante, W. Tiberti, and F. Santucci, SEAMLESS Project: Development of a Performing Secure Platform for IEEE 802.15.4 Network Applications, submitted to the ITASEC 2020 Italian Conference on Cyber-Security, 2020 https://univaq-my.sharepoint.com/:f:/g/personal/walter_tiberti_univaq_it/EihenFEx2yNDp4BcSoVRNVkBxvtlLhBhe9srfzwJbrAv3w?e=zYZ20o

[6] W. Tiberti, H. Kurunathan. "DeSTAK - A secure approach for low power deterministic networks". Work in progress, https://univaq-my.sharepoint.com/:f:/g/personal/walter_tiberti_univaq_it/EihenFEx2yNDp4BcSoVRNVkBxvtlLhBhe9srfzwJbrAv3w?e=zYZ20o

[7] D. Cassioli, A. Di Marco, L. Pomante, M. Santic, W. Tiberti. A *Model-based* porting of WSN middleware: the Agilla experience, Work in progress, `https://univaq-my.sharepoint.com/:f:/g/personal/walter_tiberti_univaq_it/EihenFEx2yNDp4BcSoVRNVkBxvtlLhBhe9srfzwJbrAv3w?e=zYZ20o`

[8] Berardinelli L., Di Marco A., Pace S., Pomante L., Tiberti W. "Energy Consumption Analysis and Design of Energy-Aware WSN Agents in fUML", Modelling Foundations and Applications, Springer International Publishing, 2015, 10.1007/978-3-319-21151-0_1

[9] Vittorio Cortellessa, Antinisca Di Marco, Daniele Di Pompeo, Francesco Gallo, Stefano Pace, Luigi Pomante, and Walter Tiberti. 2018. Energy-Driven Reconfiguration of Applications for Wireless Sensor Networks. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18). ACM, New York, NY, USA, 79-84. DOI: https://doi.org/10.1145/3185768.3186312

[10] Pugliese M., "Managing Security Issues in Advanced Applications of Wireless Sensor Networks", PhD Thesis, 2008

[11] Marchesani S., "A Middleware approach for WSN security: Cryptography and Intrusion Detection for Real-World Application", PhD Thesis, 2013

[12] L. Pomante, M. Pugliese, S. Marchesani and F. Santucci, "WINSOME: A middleware platform for the provision of secure monitoring services over Wireless Sensor Networks," 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), Sardinia, 2013, pp. 706-711. doi: 10.1109/IWCMC.2013.6583643

[13] Pugliese M., Giani A., Santucci F. (2010) Weak Process Models for Attack Detection in a Clustered Sensor Network Using Mobile Agents. In: Hailes S., Sicari S., Roussos G. (eds) Sensor Systems and Software. S-CUBE 2009. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 24. Springer, Berlin, Heidelberg

[14] M. Pugliese, L. Pomante, and F. Santucci "Secure Platform over Wireless Sensor Networks," in Applied Cryptography and Network Security, ISBN 978-953-51-0218-2, INTECH Publishers, 2012

[15] P. Giri, K. Ng and W. Phillips, "Wireless Sensor Network System for Landslide Monitoring and Warning," in IEEE Transactions on Instrumentation and Measurement, vol. 68, no. 4, pp. 1210-1220, April 2019.

[16] IEEE Standard for Low-Rate Wireless Networks," in IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011) , vol., no., pp.1-709, 22 April 2016 doi: 10.1109/IEEESTD.2016.7460875

[17] IEEE Recommended Practice for Transport of Key Management Protocol (KMP) Datagrams," in IEEE Std 802.15.9-2016 , vol., no., pp.1-74, 17 Aug. 2016 doi: 10.1109/IEEESTD.2016.7544442

[18] M Pugliese, L Pomante, F Santucci. "Agent-based scalable design of a cross-layer security framework for wireless sensor networks monitoring applications". Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International, 2009.

[19] M. Pugliese and F. Santucci, "Pair-wise network topology authenticated hybrid cryptographic keys for Wireless Sensor Networks using vector algebra," 2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, Atlanta, GA, 2008, pp. 853-859. doi: 10.1109/MAHSS.2008.4660137

[20] Marchesani S., Pomante L., Pugliese M., Santucci F. (2013) Definition and Development of a Topology-Based Cryptographic Scheme for Wireless Sensor Networks. In: Zuniga M., Dini G. (eds) Sensor Systems and Software. S-CUBE 2013. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 122. Springer, Cham

[21] Texas Instruments MSP430 Micro-controllers homepage, `http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html`

[22] Microchip homepage, `https://www.microchip.com`

[23] Texas Instrument CC2420 homepage, `http://www.ti.com/product/CC2420`

[24] RF230 product datasheet, `http://ww1.microchip.com/downloads/en/devicedoc/doc5131.pdf`

[25] T. Watteyne et al., "Industrial Wireless IP-Based Cyber –Physical Systems," in Proceedings of the IEEE, vol. 104, no. 5, pp. 1025-1038, May 2016. doi: 10.1109/JPROC.2015.2509186

[26] CRC16-CCITT, `http://srecord.sourceforge.net/crc16-ccitt.html`

[27] Transmission of IPv6 Packets over IEEE 802.15.4 Networks, `https://tools.ietf.org/html/rfc4944`

[28] The Constrained Application Protocol (CoAP), `https://tools.ietf.org/html/rfc7252`

[29] Representational state transfer, `https://en.wikipedia.org/wiki/Representational_state_transfer`

[30] TinyOS project homepage, `http://www.tinyos.net`

[31] J. Hauer, "TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2", TKN Technical Report TKN-08-003, Berlin, March 2009, URL: *http://www.tkn.tu-berlin.de/fileadmin/fg112/Papers/TKN154.pdf*

[32] Matsumoto, Makoto and Nishimura, Takuji, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation (TOMACS), vol. 8, pages 3-30, 1998

[33] Lenore Blum, Manuel Blum, and Michael Shub, "A Simple Unpredictable Pseudo-Random Number Generator", SIAM Journal on Computing, volume 15, p.364–383, may 1986

[34] C. L. Fok, G. C. Roman, and C. Lu, "Agilla: A Mobile Agent Middleware for Self-Adaptive Wireless Sensor Networks", ACM Transactions on Autonomous and Adaptive Systems, Vol. 4, No. 3, Article 16, 2009

[35] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," 2008 International Conference on Information Processing in Sensor Networks (ipsn 2008), St. Louis, MO, 2008, pp. 245-256.

[36] Contiki-ng homepage, `https://www.contiki-ng.org/`

[37] Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, Matthias Wählisch, RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT, IEEE Internet of Things Journal, Vol. 5, No. 6, pp. 4428-4440, December 2018.

[38] TinyDB homepage, `http://telegraph.cs.berkeley.edu/tinydb`

[39] `https://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes`

[40] Hadim and N. Mohamed, "Middleware: middleware challenges and approaches for wireless sensor networks", in IEEE Distributed Systems Online, vol. 7, no. 3, pp. 1-1, March 2006.

[41] Levis, P. and Culler, D., "Mate: A Tiny Virtual Machine for Sensor Networks", Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), (2002).

[42] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho and M. A. Perillo, "Middleware to support sensor network applications", in IEEE Network, vol. 18, no. 1, pp. 6-14, Jan/Feb 2004.

[43] Agilla ISA: `http://mobilab.wustl.edu/projects/agilla/isa.html`

[44] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`

[45] Morris J. Dworkin. 2007. SP 800-38d. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (Gcm) and GMAC. Technical Report. NIST, Gaithersburg, MD, United States.

[46] Daniel J. Bernstein, Salsa20 and ChaCha20 home page, `https://cr.yp.to/hash.html`

[47] RFC 2631 – Diffie–Hellman Key Agreement Method. E. Rescorla. June 1999. `https://tools.ietf.org/html/rfc2631`

[48] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 2 (February 1978), 120-126. DOI=http://dx.doi.org/10.1145/359340.359342

[49] Taher El Gamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. In Proceedings of CRYPTO 84 on Advances in cryptology, G R Blakley and David Chaum (Eds.). Springer-Verlag New York, Inc., New York, NY, USA, 10-18.

[50] PKCS #7: Cryptographic Message Syntax, `https://tools.ietf.org/html/rfc2315`

[51] US Secure Hash Algorithms (SHA and HMAC-SHA), `https://tools.ietf.org/html/rfc4634`

[52] FIPS PUB 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`

[53] N. Koblitz, Elliptic curve cryptosystems, Mathematics of Computation, 48 (1987), pp. 203-209

[54] R. Daidone, "Experimental evaluations of security impact on IEEE 802.15.4 networks," 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, Lucca, 2011, pp. 1-2. doi: 10.1109/WoWMoM.2011.5986151

[55] Walter Tiberti, TAKS fast key component generator,

[56] P. Choi, J. Kong and D. K. Kim, "Analysis of hardware modular inversion modules for elliptic curve cryptography," 2015 International SoC Design Conference (ISOCC), Gyungju, 2015, pp. 313-314. doi: 10.1109/ISOCC.2015.7401713

[57] Imran, Malik, Imran Shafi, Atif Raza Jafri and Muhammad H. Rashid. "Hardware design and implementation of ECC based crypto processor for low-area-applications on FPGA." 2017 International Conference on Open Source Systems & Technologies (ICOSST) (2017): 54-59.

[58] R. Shahid, T. Winograd and K. Gaj, "A Generic Approach to the Development of Coprocessors for Elliptic Curve Cryptosystems," 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, 2017, pp. 158-167. doi: 10.1109/IPDPSW.2017.166

[59] Su-Wen Yi, Wei Li, Zi-Bin Dai and Jun-Wei Liu, "A compact and efficient architecture for elliptic curve cryptographic processor," 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Hangzhou, 2016, pp. 1276-1280. doi: 10.1109/ICSICT.2016.7998714

[60] P. L. Montgomery, "Modular Multiplication Without Trial Division", Mathematics of Computation, vol. 44, num. 170, 1985

[61] GHDL - VHDL compiler and simulator, `http://ghdl.free.fr/`

[62] Xilinx Vivado Design Suite, `https://www.xilinx.com/products/design-tools/vivado.html`

[63] Walter Tiberti GitHub page, `https://www.github.com/wtiberti`

[64] PBKDF2 Key Derivation Function, `https://en.wikipedia.org/wiki/PBKDF2`

[65] D.J. Bernstein, Poly1305 message authentication code, `https://en.wikipedia.org/wiki/Poly1305`

[66] Vivado High Level Synthesis, `https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`

[67] OpenZB homepage, `http://www.open-zb.net/`

[68] Routing Requirements for Urban Low-Power and Lossy Networks, `https://tools.ietf.org/html/rfc5548`

[69] TinyRPL homepage, `http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyRPL`

[70] `https://www.silabs.com/community/blog.entry.html/2016/05/27/iot_security_part7-UeMh`

[71] Hasse, Helmut (1936), "Zur Theorie der abstrakten elliptischen Funktionenkörper. I, II & III", Crelle's Journal, 1936 (175), doi:10.1515/crll.1936.175.193, ISSN 0075-4102, Zbl 0014.14903

[72] A. Karatsuba and Yu Ofman, Multiplication of Many-Digital Numbers by Automatic Computers. Doklady Akad. Nauk SSSR Vol. 145 (1962), pp. 293–294

[73] ZigBee Alliance Home, `https://zigbeealliance.org`

[74] Fan X., F. Susan, W. Long, S. Li, "Security Analysis of ZigBee", 2017

[75] Ended Yüksel, "Analysing ZigBee Key Establishment Protocols", arXiv:1205.6678, May 2012

[76] Law L., Menezes A., Qu M., Solinas J., Vanstone S. : "An Efficient Protocol for Authenticated Key Agreement", Designs, Codes and Cryptography (2003) 28: 119. https://doi.org/10.1023/A:1022595222606

[77] SEC 2: Recommended Elliptic Curve Domain Parameters, `http://www.secg.org`

[78] SafeCurves: choosing safe curves for elliptic-curve cryptography, `https://safecurves.cr.yp.to`

[79] T. Itoh and S. Tsujii, A fast algorithm for computing multiplicative inverses in gf(2m) using normal bases, Inf. Comput., 78 (1988), pp. 171–177.

[80] Rezai, Abdalhossein & Keshavarzi, Parviz. (2012). CCS Representation: A new non-adjacent form and its application in ECC. Journal Basic Applied Scienctific Research. 2. 4577-4586.

[81] CBCMAC description, `https://cryptography.fandom.com/wiki/CBC-MAC`

[82] Brincat, Karl and Mitchell, Chris J., "New CBC-MAC Forgery Attacks", in Information Security and Privacy, 2001, Springer, ISBN 978-3-540-47719-8

[83] IEEE Standard Specifications for Public-Key Cryptography," in IEEE Std 1363-2000 , vol., no., pp.1-228, 29 Aug. 2000 doi: 10.1109/IEEESTD.2000.92292

[84] S. Scaccialepre, G. Piro, G. Boggia, G. Bianchi. "Public Key Authentication and Key Agreement in IoT Devices With Minimal Airtime Consumption", in IEEE Embedded Systems Letters, vol. 9, NO. 1, March 2017

[85] D. R. L. Brown, R. Gallant, and S. A. Vanstone, "Provably secure implicit certificate schemes," in Proc. Finan. Cryptogr., Grand Cayman, Cayman Islands, 2001, pp. 156–165.

[86] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang. High-speed high-security signatures. Journal of Cryptographic Engineering 2 (2012), 77–89. Document ID: a1a62a2f76d23f65d622484ddd09caf8. URL: https://cr.yp.to/papers.html#ed25519. Date: 2011.09.26

[87] H. Nunoo-Mensah, K. O. Boateng, and J. D. Gadze. Tamper-aware authentication framework for wireless sensor networks. IET Wireless Sensor Systems, 7(3):73–81, 2017.

[88] R. Tahir and K. McDonald-Maier. Improving resilience against node capture attacks in wireless sensor networks using icmetrics. In 2012 Third International Conference on Emerging Security Technologies, pages 127–130, Sep. 2012.

[89] X. Jin, P. Putthapipat, D. Pan, N. Pissinou, and S. K. Makki. Unpredictable software-based attestation solution for node compromise detection in mobile wsn. In 2010 IEEE Globecom Workshops, pages 2059–2064, Dec 2010.

[90] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: software-based attestation for embedded devices. In IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004, pages 272–282, May 2004.

[91] Manyam Thaile and OBV Ramanaiah. Node compromise detection based on parameter grouping in wireless sensor networks, 2016.

[92] OpenDSME project homepage, http://opendsme.org/

[93] Omnet++ network simulator project homepage, https://omnetpp.org/

[94] L. Corradetti, D. Gregori,S. Marchesani, L. Pomante,M. Santic and W. Tiberti. "A renovated mobile agents middleware for WSN porting of Agilla to the TinyOS 2.x platform".IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow, RTSI 2016, Bologna, Italy, pp.1–5.

[95] Agilla2 repository, https://github.com/luigi-pomante/Agilla2

[96] Celia Chen, "How to measure and Estimate Software Maintainability for Open Source Projects?", url: http://csse.usc.edu/new/wp-content/uploads/2017/01/How-to-Measure-and-Estimate-Software-Maintainability-for-Open-Source-Projects-.pdf

[97] P. Pop, D. Scholle, H. Hansson, G. Widforss and M. Rosqvist, "The SafeCOP ECSEL Project: Safe Cooperating Cyber-Physical Systems Using Wireless Communication," 2016 Euromicro Conference on Digital System Design (DSD), Limassol, 2016, pp. 532-538. doi: 10.1109/DSD.2016.25

[98] G. Agosta et al., "V2I Cooperation for Traffic Management with SafeCop," 2016 Euromicro Conference on Digital System Design (DSD), Limassol, 2016, pp. 621-627. doi: 10.1109/DSD.2016.18

[99] Memsic IRIS WSN platform datasheet, `http://www.memsic.com/userfiles/files/User-Manuals/iris-oem-edition-hardware-ref-manual-7430-0549-02.pdf`

[100] Lattice Semiconductor homepage, `http://www.latticesemi.com`

[101] Microsemi homepage, `https://www.microsemi.com`

[102] TKN154 physical driver for AT86RF230, `https://github.com/Loki88/TKN154-Iris`

[103] D. Cassioli, A. Di Marco, F. Gallo, S. Pace, L. Pomante and C. Rinaldi, "VISION: Video-oriented UWB-based intelligent ubiquitous sensing," 2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE), Eilat, 2016, pp. 1-5. doi: 10.1109/ICSEE.2016.7806102

[104] T. M. Mubarak, S. A. Sattar, A. Rao and M. Sajitha, "Energy efficient intrusion detection in three dimensional wireless sensor networks," 2010 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, 2010, pp. 1-4.

[105] M. Sheikhan and H. Bostani, "A hybrid intrusion detection architecture for Internet of things," 2016 8th International Symposium on Telecommunications (IST), Tehran, 2016, pp. 601-606

[106] Z. Sun, Y. Xu, G. Liang and Z. Zhou, "An Intrusion Detection Model for Wireless Sensor Networks With an Improved V-Detector Algorithm," in IEEE Sensors Journal, vol. 18, no. 5, pp. 1971-1984, 1 March1, 2018.

[107] T. Maphatsoe and M. Masinde, "Asymptotic Analysis of A Fuzzy Based Intrusion Detection System For Zigbee," 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), Plaine Magnien, 2018, pp. 1-8.

[108] Alrajeh, Nabil & Khan, Salim & Shams, Bilal. (2013). Intrusion Detection Systems in Wireless Sensor Networks: A Review. International Journal of Distributed Sensor Networks. 2013. 10.1155/2013/167575.

[109] OpenSSL project Homepage, `https://www.openssl.org`

[110] Remote Code Execution (definition), `https://en.wikipedia.org/wiki/Arbitrary_code_execution`

[111] MQTT protocol, `http://mqtt.org`

[112] P. R. Chandre, P. N. Mahalle and G. R. Shinde, "Machine Learning Based Novel Approach for Intrusion Detection and Prevention System: A Tool Based Verification," 2018 IEEE Global Conference on Wireless Computing and Networking (GCWCN), Lonavala, India, 2018, pp. 135-140. doi: 10.1109/GCWCN.2018.8668618

[113] Veeramreddy Jyothsna and Koneti Munivara Prasad, Anomaly-Based Intrusion Detection System, DOI: 10.5772/intechopen.82287

[114] TinyOS TEPs, `http://tinyos.stanford.edu/tinyos-wiki/index.php/TEPs`

[115] SafeCOP project homepage, `http://www.safecop.eu`

[116] M. H. Eldefrawy, M. K. Khan and K. Alghathbar, "A key agreement algorithm with rekeying for wireless sensor networks using public key cryptography," 2010 International Conference on Anti-Counterfeiting, Security and Identification, Chengdu, 2010, pp. 1-6.

[117] H. Ghasemzadeh, M. R. Aref and A. Payandeh, "A novel and low-energy PKC-based key agreement protocol for WSNs," 2013 10th International ISC Conference on Information Security and Cryptology (ISCISC), Yazd, 2013, pp. 1-6.

[118] R. B. Gandara, G. Wang and D. N. Utama, "Hybrid Cryptography on Wireless Sensor Network: A Systematic Literature Review," 2018 International Conference on Information Management and Technology (ICIMTech), Jakarta, 2018, pp. 241-245. doi: 10.1109/ICIMTech.2018.8528147

[119] A. Bhave and S. R. Jajoo, "Secure communication in Wireless Sensor Networks using hybrid encryption scheme and cooperative diversity technique," 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, 2015, pp. 1-6. doi: 10.1109/ISCO.2015.7282235

[120] Y. Alkady, M. I. Habib and R. Y. Rizk, "A new security protocol using hybrid cryptography algorithms," 2013 9th International Computer Engineering Conference (ICENCO), Giza, 2013, pp. 109-115. doi: 10.1109/ICENCO.2013.6736485

[121] L. Harn and C. Hsu, "Predistribution Scheme for Establishing Group Keys in Wireless Sensor Networks," in IEEE Sensors Journal, vol. 15, no. 9, pp. 5103-5108, Sept. 2015. doi: 10.1109/JSEN.2015.2429582

[122] (multiple authors), "Safety and Security in V2I Traffic Management: The SafeCOP Approach", Work in progress, `https://univaq-my.sharepoint.com/:f:/g/personal/walter_tiberti_univaq_it/EihenFEx2yNDp4BcSoVRNVkBxvtlLhBhe9srfzwJbrAv3w?e=zYZ20o`

[123] Rasouli, Jafar & Motamedi, Ahmad & Baseri, Mohamad & Parsa, Mahshad. (2019). A Reliable Communication Model Based on IEEE802.15.4 for WSANs in Smart Grids. 10.5772/intechopen.84288.

[124] Afzaal, Hamra & Iqbal, Zafar & Saeed, Tahreem & Zafar, Nazir. (2017). Battlefield surveillance formalism using WSANs. 1-6. 10.1109/ICEE.2017.7893437.

[125] Fahmy, Hossam. (2016). WSNs Applications. 10.1007/978-981-10-0412-4_3.

[126] Zhao, Jingjing & Ge, Meng & Yang, Yun & Zhang, Lifeng. (2017). A Brief Review of WSN Applications. 4-7. 10.12792/iciae2017.004.

[127] L. Law, A. Mezenes, M. Qu, S. Vanstone, "An Efficient Protocol for Authenticated Key Agreement", Designs, Codes and Cryptography n.28, 2003, 10.1023/A:1022595222606

[128] Yüksel, Ender, Hanne Riis Nielson, and Flemming Nielson. "Zigbee-2007 security essentials." Proc. 13th Nordic Workshop on Secure IT-systems. 2008.