Bernard Candaele · Dimitrios Soudris
Iraklis Anagnostopoulos *Editors*

Trusted Computing for Embedded Systems

Springer

Trusted Computing for Embedded Systems

Bernard Candaele • Dimitrios Soudris
Iraklis Anagnostopoulos
Editors

Trusted Computing for Embedded Systems

Springer

Editors
Bernard Candaele
Thales, Communications
Colombes, France

Iraklis Anagnostopoulos
Department of Computer Science
National Technical University of Athens
Athens, Greece

Dimitrios Soudris
Department of Computer Science
National Technical University of Athens
Athens, Greece

ISBN 978-3-319-09419-9 ISBN 978-3-319-09420-5 (eBook)
DOI 10.1007/978-3-319-09420-5
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014955601

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Semiconductors are a technology shaping our life in a transparent way. Every day, we touch upon hundreds of semiconductor empowered objects and services at home, at work, when travelling, when getting healthy or simply when having fun. Semiconductors are to the knowledge-based society what steel and coal have been to the industrial, and what grains have been to the agrarian society.

The unparalleled capability to reduce the unit cost (about one million times in 25 years!) by miniaturisation and functional diversification requires, paradoxically, ever increasing upfront investments. Industrial companies, institutional and academic researchers, but also public entities are engaged in an intense global competition with strategic implications.

The ENIAC Joint Undertaking takes the European public-private partnerships to the next level bringing together industrial, institutional and academic R&D actors with the national public authorities and the European Commission in a coherent, transparent way. The goal is to make a recognisable contribution towards a globally successful and sustainable European nano-electronics industry by allocating the approximate 3 billion Euros of the programme to the most impactful R&D topics. The exceptionally high quality of our stakeholders and the strong commitment of our dedicated team are the ingredients that will enable the ENIAC Joint Undertaking to pursue its vision and reach its goals and objectives (Fig. 1).

The ENIAC Joint Undertaking contributes to the implementation of the Seventh Framework Programme and the theme 'Information and Communication Technologies' of the Specific Programme 'Cooperation'.

In particular:

- Defines and implements a Research Agenda for the development of key competences for nano-electronics across different application areas in order to strengthen European competitiveness and sustainability and to stimulate the emergence of new markets and societal applications
- Awards funding to participants in selected projects following calls for competitive proposals

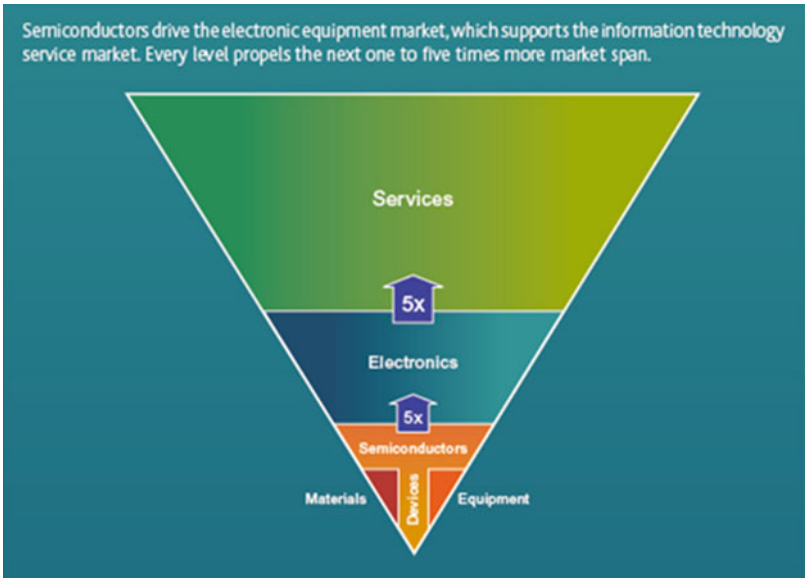


Fig. 1 ENIAC Joint Undertaking field of interest

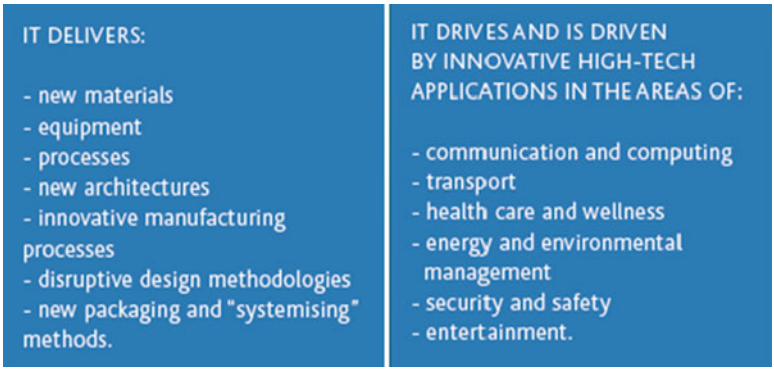


Fig. 2 ENIAC joint undertaking main objectives

- Mobilizes and pools the European Union, national and private efforts increasing overall R&D investments and fostering collaboration between the public and private sectors
- Works towards achieving synergy and coordination of European R&D efforts promotes the involvement of SMEs in its activities (Fig. 2)

The ENIAC Joint Undertaking addresses two main objectives:

- Enhancing the further integration and miniaturisation of devices
- Increasing their functionalities

Fig. 3 The bodies of the ENIAC joint undertaking

and its members are:

- The European Union, represented by the Commission
- Member States and associated countries subscribing to the objectives and accepting the Statutes
- AENEAS, an association acting as the representative of companies and other R&D actors operating in the field of nano-electronics in Europe

The following entities may also become Members of the ENIAC Joint Undertaking upon subscribing to the objectives and assuming all obligations of membership:

- Any third country pursuing R&D policies or programmes in the area of nano-electronics
- Any other legal entity capable of making a substantial financial contribution to the achievement of the objectives of the ENIAC Joint Undertaking

The bodies of the ENIAC Joint Undertaking are depicted in Fig. 3.

- Governing Board
  - Approves Multi Annual Strategic Plan
  - Approves Annual Implementation Plan
  - Approves Rules of Procedure

- Executive Director
  - Daily operation
  - Approves Annual Implementation Plan
- Public Authorities Board
  - Approves Annual Work Plan, calls for proposals
- Industry and Research Committee
  - Elaborates Multi Annual Strategic Plan

Regarding the financing model, SMEs, research institutes and universities may have different funding thresholds for the national grants depending on the rules applied by each State.

At the end of its life span, ENIAC Joint Undertaking selected for funding 64 projects with total eligible cost of 2.9 billion Euros, with 1,384 participants from 24 Member States and associated countries.

The ENIAC Joint Undertaking established itself as the leading mechanism in implementing Key Enabling Technology policies, by launching 14 pilot line projects that shifted the center of interest closer to the innovation with considerable economic impact.

Brussels, Belgium Andreas Wild
June 2014

Preface

Computing systems have a tremendous impact on everyday life in all domains, from the internet to consumer electronics, transportation to manufacturing, medicine, energy, and scientific computing. Also, the traditional perspective of single-purpose and single-core embedded systems and devices is rapidly changing as increased computing performance and functionality is being added by industry leaders. Convergence, both in hardware and software platforms, is rampant throughout the industry with desktop processors and embedded processors merging together. These hardware changes are also driven by application changes. Future applications and everyday solutions, such as smart grids for electricity network, smart low energy controlled home appliance, environmental or infrastructure sensor networks, communication networks and wireless communications, will start to become dominant the next years.

However, management of trusted components and a number of secure technologies need to be developed and put in place in order to make both hardware and software solutions smarter and more secure. The interest is so big that industrial companies have started to define, develop and validate trust hardware and firmware mechanisms applicable to embedded devices and use them as security anchors within related embedded platforms.

A project aimed at analyzing security problems and at suggesting solutions for trusted embedded systems was the ENIAC project TOISE with participants from seven European countries. The first objective of TOISE, with regards to the objective energy efficiency, was to investigate and implement secure solutions for the design of smart-grid applications and their deployment in large-scale networks and systems. The second objective of TOISE, with regards to the security of communications, was to investigate and implement secure wireless sensor networks, to address secure authentication devices and to study and implement new generation of trusted portable devices as well secure storage in memory. Last, the third objective was to develop a new generation of tokens (based on several form factors) that will demonstrate optimal cost through enforced privacy management and secure channel establishment.

This book describes, but it is not limited to, the results of the TOISE project. Therefore, the purpose of this book is threefold. Firstly, to be used as an undergraduate or graduate level textbook for introduction to trusted embedded systems providing students and practicing engineers with the fundamentals as well as details in the many facets of security problems. Secondly, to be used as reference for researchers in the field. Thirdly, to be used as a guide for professionals in analyzing and designing state-of-the-art trusted embedded systems.

The book consists of three parts. The first is an introductory chapter presenting information about the existing solutions and programming interfaces in trusted domain. The second presents four real applications-use cases which depict the need for designing trusted embedded systems. Last, based on the second part, the third presents the building blocks on which the presented applications-use cases were built. In other words, the third part shows and analyzes the core technologies used in the use cases.

We would like to extend our gratitude to all members of the TOISE team for an exciting project collaboration and many inspiring discussions. We want to thank the authors for the contributions and the hard work on the chapters of this book making it a concise and representative summary of three years of research and development. Furthermore, we would like to express our appreciation to the project reviewer's comments and feedback that were always insightful and to the point, and that greatly helped us to stay focused, to increase our efforts, and to keep our overall objectives in mind. We would also like to thank our project officer (Fabrizio Martone) for his professional and sensible handling of TOISE, and last but not least, we are very grateful to the team of Springer who has enthusiastically supported this book from the very beginning, very professionally transformed the material into a high quality publication, and kept patience and support when the delivery of the material were behind schedule. Most importantly we hope that you, the reader, enjoy reading this book and that it triggers your inspiration and many new ideas.

Paris, France — Bernard Candaele
Athens, Greece — Dimitrios Soudris
June 2013 — Iraklis Anagnostopoulos

Contents

Contributors

Asad Ali Gemalto, Austin, TX, USA

Bibiana Lorente Alvarez CNM-IMB (CSIC), Centro Nacional de Microelectrónica – Instituto de Microelectrónica de Barcelona, Consejo Superior de Investigaciones Científicas. Campus de la Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain

Iraklis Anagnostopoulos School of Electrical and Computer Engineering, National Technical University of Athens, Athina, Greece

Christos Andrikos School of Electrical and Computer Engineering, National Technical University of Athens, Athina, Greece

Alessandro Barenghi Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milan, Italy

Luca Breveglieri Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milan, Italy

Luis Andres Cardona Cardona CNM-IMB (CSIC), Centro Nacional de Microelectrónica – Instituto de Microelectrónica de Barcelona, Consejo Superior de Investigaciones Científicas. Campus de la Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain

Zouha Cherif Telecom ParisTech, Paris, France

Jean-Luc Danger Telecom ParisTech, Paris, France

Secure-IC, Paris, France

Álvaro Díaz University of Cantabria, Santander, Spain

Sylvain Duvillard Magillem, Paris, France

Jacques J.A. Fournier CEA, Gardanne, France

Julien Francq Airbus Defence and Space – CyberSecurity, Elancourt Cedex, France

Mariagrazia Fugini Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milan, Italy

Jon Geater Trustonic Ltd., Cambridge, UK

Michael Gissing Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Graz, Austria

Javier González University of Cantabria, Santander, Spain

Sylvain Guilley Telecom ParisTech, Institut Mines-Telecom, CNRS LTCI, Department Comelec, Paris Cedex 13, France

Annelie Heuser Telecom ParisTech, Institut Mines-Telecom, CNRS LTCI, Department Comelec, Paris Cedex, France

Theodore W. Hong University of Cambridge Computer Laboratory, Cambridge, UK

Bernard Kasser STMicroelectronics, Rousset, France

Ioannis Koutras School of Electrical and Computer Engineering, National Technical University of Athens, Athina, Greece

Apostolos Leventis Hellenic Aerospace Industry Tanagra, Schimatari, Greece

Florent Lozac'h Telecom ParisTech, Paris, France

Albert Martinez STMicroelectronics, Rousset, France

Gerald Maunier Gemalto, La Ciotat Cedex, France

Mohammad Nauman Computer Science Research and Development Unit, Peshawar, Pakistan

Philippe Nguyen Secure-IC, Rennes, France

Andrea Palomba Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milan, Italy

Konstantinos Papadopoulos Hellenic Aerospace Industry Tanagra, Schimatari, Greece

Geoffrey Parlier Airbus Defence and Space – CyberSecurity, Elancourt Cedex, France

Gerardo Pelosi Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Milan, Italy

Carles Ferrer Ramis CNM-IMB (CSIC), Centro Nacional de Microelectrónica – Instituto de Microelectrónica de Barcelona, Consejo Superior de Investigaciones Científicas and Department de Microelectrónica i Sistemes Electrònics, Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain

Guillaume Reymond CEA, Gardanne, France

Juan Rico TST, Santander, Spain

Olivier Rioul Telecom ParisTech, Institut Mines-Telecom, CNRS LTCI, Department Comelec, Paris Cedex 13, France

Pablo Sánchez University of Cantabria, Santander, Spain

Juan Sancho TST, Santander, Spain

Dimitrios Soudris School of Electrical and Computer Engineering, National Technical University of Athens, Athina, Greece

Ronald Toegl Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Graz, Austria

François Tuot Gemalto, Meudon Cedex, France

Emmanuel Vaumorin Magillem, Paris, France

Thomas Winkler Pervasive Computing Group/Institute of Networked and Embedded Systems (NES), Alpen-Adria Universitaet Klagenfurt, Klagenfurt, Austria

Johannes Winter Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Graz, Austria

Acronyms

ACK	Acknowledgment
AES	Advanced Encryption Standard
AIK	Attestation Identity Key
API	Application programming interface
ASIC	Application specific integrated circuit
BS	Base station
CA	Certificate authorities
CH	Cluster head
CIP	Critical infrastructure protection
CPUF	Controlled PUF
CRC	Cyclic redundancy check
CRP	Challenge response pair
DES	Data Encryption Standard
DESX	Data Encryption Standard XORed
DFA	Differential fault analysis
DMM	Dynamic memory management
DoS	Denial of service
DRM	Digital right management
DSO	Distribution service operator
DT	Decision tree
EC	Executive committee
ECC	Error correcting code
EG	Expert group
ESP	Energy service provider
FAR	False acceptance rate
FC	Final customer
FIQ	Fast interrupt request
FPGA	Field programmable gate array
FRR	False rejection rate
FSA	Fault sensitivity analysis
FSMC	Flexible static memory controller

HD	Hamming distance
HM	Hard macro
HMAC	Hashed message authentication code
HW	Hardware
IC	Integrated circuit
ICT	Information and communication technologies
IDE	Integrated development environment
IIA	Inter-class information analysis
IP	Intellectual property
IRQ	Interrupt request
ISS	Instruction set simulator
IT	Information technologies
JCP	Java community process
JIT	Just in time
JNI	Java native interface
JRE	Java runtime environment
JSR	Java specification request
JTAG	Joint Test Action Group
JVM	Java virtual machine
LAB	Logic array blocks
LFSR	Linear feedback shift register
LPC	Low pin count
LPUF	Loop PUF
LPWSN	Low power wireless sensor network
LUT	Look-up table
MAC	Medium access control
MCU	Master control unit
MD5	Message Digest 5 Algorithm
MIA	Mutual information analysis
MMU	Memory management unit
MoC	Match-on-card
MPSoC	Multi processor system-on-chip
MPU	Memory protection unit
MUX	Multiplexer
NC	Network coordinator
NS	Non secure
NWd	Normal world
OAEP	Optimal asymmetric encryption padding
OATH	Open authentication
OS	Operating system
OTAP	Over the air programming
OTP	One time password
PC	Personal computer
PCR	Platform configuration register
PDA	Personal digital assistant

PDF	Probability density function
PIN	Personal identification number
PK	Public key
PKCS	Public-Key Cryptography Standards
PKI	Public key infrastructure
PMF	Probability mass function
PRIME	PoweRline Intelligent Metering Evolution
PUF	Physically unclonable function
PUFKY	Key generator using a PUF
QoS	Quality-of-service
RAM	Random access memory
RF	Radio frequency
RFID	Radio frequency identification device
RI	Reference implementation
ROPUF	Ring oscillator PUF
RTL	Register transfer level
RTOS	Real time operating system
SHA	Secure hash algorithm
SHD	Sum of hamming distance
SM	Success metric
SML	Stored measurement log
SMS	Short message service
SN	Sensor node
SNR	Signal-to-noise ratio
SOAP	Simple object access protocol
SoC	System-on-chip
SR	Success rate
SSL	Secure sockets layer
SW	Software
SWd	Secure world
TBS	TPM base services
TC	Trusted computing
TCB	Trusted computing base
TCG	Trusted Computing Group
TCK	Technology compatibility kit
TCS	TSS core services
TDDL	Trusted device driver library
TDES	Triple Data Encryption Standard
TEE	Trusted execution environment
TIS	TPM interface specification
TLS	Transport layer security
TOCTTOU	Time of check to time of use
TPM	Trusted platform module
TSP	TSS service provider
TSS	Trusted software stack

UART	Universal asynchronous receiver transmitter
UUID	Universally unique identifier
VHDL	VHSIC Hardware Description Language
VM	Virtual machine
VPN	Virtual private network
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless local area network
WSDL	Web Services Description Language
WSN	Wireless sensor network

Part I
Introduction

Chapter 1
Programming Interfaces for the TPM

Ronald Toegl, Thomas Winkler, Mohammad Nauman, Theodore W. Hong, Johannes Winter, and Michael Gissing

Abstract The paradigm of Trusted Computing promises a new approach to improve the security of embedded and mobile systems. The core functionality, based on a hardware component known as Trusted Platform Module (TPM), is widely available. However, integration and application in embedded systems remains limited at present, simply because of the extremely steep learning curve involved in using the programmer-facing interfaces. In this chapter, we describe the current state of the Trusted Computing Group's software architecture and present previous approaches to improve usability. We report on a novel design of a high-level API for Trusted Computing for Java which has been optimized for ease-of-use and clear abstraction of Trusted Computing concepts. We derive requirements and design goals and outline the API design. Finally, we show the application and benchmarks in embedded systems. The result of this effort has been standardized as Java Specification Request 321.

1.1 Introduction

Embedded Systems take many forms, such as mobile phones, industrial control systems, network devices, sensor nodes, and smart cards. Having become nearly ubiquitous, the world Embedded Systems market exceeds 100 billion USD [18]. Often, sensitive information is created, accessed, manipulated, stored, and communicated

R. Toegl (✉) • J. Winter • M. Gissing
Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
e-mail: rtoegl@iaik.tugraz.at

T. Winkler
Pervasive Computing Group/Institute of Networked and Embedded Systems (NES),
Alpen-Adria Universitaet Klagenfurt, Lakeside Park B02b, A-9020 Klagenfurt, Austria

M. Nauman
Computer Science Research and Development Unit, Peshawar, Pakistan

T.W. Hong
University of Cambridge Computer Laboratory, William Gates Building,
15 J.J. Thomson Ave., Cambridge CB3 0FD, UK

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_1

on such Embedded Systems. Thus, security needs to be considered throughout the design process [50], including hardware design and software development. Specifically, in the Trusted Computing approach, security is bootstrapped from a small dedicated piece of secure hardware, the Trusted Platform Module (TPM).

While most the major computer manufacturers are shipping servers, desktop and notebook computers containing TPMs with several hundreds of million of machines can be assumed to provide this hardware device [58], its application to Embedded Systems has only been limited. Major obstacles to the development of Trusted Computing enabled software have been the high complexity of the specification of the software stack that is used to manage the TPM and limited support for programming languages that support different hardware platforms [57, 60].

In particular, there has been insufficient support for platform-independent runtime environments like .NET [40], Android or Java. Such environments are particularly useful for implementing modern security solutions on heterogeneous platforms. For instance, several billions of devices support Java, and Oracle claims [43] that the Java developer community, with nine million members, is the largest of its kind. It is of little surprise that there have been a number of attempts to provide TPM libraries that target such a programming languages. Yet there was a lack of an established, generally-accepted Application Programming Interface (API) for TPM access.

In this chapter, which extends on [71], we describe the design of a *high-level* Java API for Trusted Computing, which has been published as an official Java *standard* [68]. Our goal in designing this API is to provide a simpler, high-level interface to the TPM while still adhering to the concepts and standards defined by the Trusted Computing Group. Benchmark results show the suitability in Embedded Systems using the TPM.

1.2 Trusted Computing in the Java Environment

1.2.1 Java for Embedded Systems

At the application layer, the Java programming environment has seen a broad adoption ranging from large-scale business applications hosted in dedicated data centers to resource constrained environments as found in mobile phones or Personal Digital Assistants (PDAs), set-top boxes, industrial control and even smart cards. Java program code [21] is not compiled to native machine code but to a special form of intermediate code, called byte code. This byte code is then executed by a virtual machine (VM) [35] called the Java VM. This characteristic makes Java an excellent choice for development aiming at heterogeneous environments. In contrast to conventional programming languages such as C or C++, Java is equipped with inherent security features supporting the development of more secure software. Among those features are automatic array-bounds checking, garbage collection and access control mechanisms. Additional aspects that distinguish Java from other

environments are code-signing mechanisms and the verification of byte code when it is loaded. The class-loading mechanism separates privileged code and creates a sandbox for remotely fetched classes [19].

For Embedded Systems development, Java offers a number of advantages [65]. Its hardware-independent architecture hides specifics of the hardware and operating system, as it is abstracted through the platform-specific, often optimized implementation of the Java VM. The rich libraries of the Java Runtime Environment (JRE) offer much more features than operating system APIs. Java also eases the creation of network inter-operable embedded systems and simplifies the software development. For very small systems, Java ME offers small-footprint configurations of Java, albeit with limited functionality. The full-featured Java SE is found on PCs and medium-to-large (i.e. 32 MB of RAM or more) Embedded Systems.

With Java being a key component, Android [20] is now the first choice for mobile smart-phones. Based on a Linux kernel, it offers a broad application library framework and the Dalvik virtual machine with just-in-time compilation which is optimized for resource-restrained devices. In addition, Android is fully source-code compatible with Java. As of 2013, Android is the most widely used, off-the-shelf operating system in Embedded Systems [77].

Here also, Trusted Computing is very promising to further improve security. While generic cryptography is well supported with the Java Security Architecture, there is currently no established standard API for Trusted Computing available. Still, a large number of Java-based use cases have been demonstrated for Trusted Computing, using several existing approaches for Trusted Computing integration in Java.

1.3 TCG Software Architecture

1.3.1 The TCG Software Stack

The TPM design is intended to allow for cost effective implementations on hardware architectures with restricted resources, such as smart card platforms. Consequently, the functionality of the TPM is restricted to what is offered by its API. The TPM is not able to execute custom code, and even most of the features offered require auxiliary functionality implemented in software.

Of the TCG standards, the *TCG Software Stack (TSS)* [73] is responsible to access and manage the TPM and also to provide a programming interface for TC applications. The standard document is accompanied with C header and Web Services Description Language (WSDL) interface definition files. The target language for the standard is the C programming language [26].

The TSS offers a set of function calls that help perform a number of operations. These functionalities cover the setup and *administration* of the TPM, such as taking ownership, the setting of configurations or the querying of properties. With regards to the chain of trust, it is the task of the TSS to record a *Stored Measurement Log*

(SML) for tracing the measurements that led to the current PCR values. The life cycle of *cryptographic keys* is also controlled through the TSS, starting with the creation of public-private key-pairs. The limited resources of the TPM necessitate external, encrypted storage of the cryptographic material, either at run-time by swapping out keys from limited hardware key slots into main memory or filing in *persistent storage* on the hard disk. The TSS also supports different mechanisms of *key certification* and the backup to other TPMs in a protocol called *migration*.

The TSS is also specified to enable Identity Management. To prevent privacy issues by correlation of the reuse of the same unique key pair for different services, the unique Endorsement Key cannot be used directly. Instead, *Attestation Identity Keys* (AIKs) are created in a process that involves the TPM and a trusted third party called a PrivacyCA. The TSS is the entity that collects all required information and certificates to assemble the appropriate data structures for communication between the local TPM chip and the remote PrivacyCA service. In [46], we outline this protocol in more detail. The alternative protocol of Direct Anonymous Attestation [9], is based on group signatures. This protocol allows a TPM to proof that it is within a group of TCG-compliant TPMs without revealing which member it is. However, the scheme described through the TPM and TSS specifications has been found to be complex to use and very slow in practical implementations [14].

Data can be encrypted by the TPM mainly using two mechanisms, binding and sealing. *Binding*, which is done in software, potentially even on a remote host, is the encryption of a limited amount of data with a public RSA key using either PKCS #1 version 1.5 [30] or OAEP [6] paddings. If the corresponding private key is unique and held by a TPM this implies that only this TPM can decrypt the data. In an even stronger mechanism called *sealing*, the encryption is performed on-chip incorporating a unique secret and a set of PCRs. Sealed data can only be unsealed by precisely the same TPM in the desired PCR configuration.

The TSS also provides interfaces to *sign* user data using TPM-protected keys, which were generated with type information that allows them to perform RSA-signature operations. AIKs are even more restricted and can only be used to sign TPM internal data structures. The most prominent example is the *Quote* operation where a set of PCRs is signed to report the current platform state. Further useful interfaces of the TSS provide access to the random number generator, a tick counter that can potentially be correlated with real-time and a monotonic counter mechanism.

From a software engineering perspective, the TSS specification follows a layered architecture shown in Fig. 1.1. Just below the TSS, and not part of it, is the TPM driver. The TPM driver can be either vendor specific or follow the TPM Interface Specification (TIS) standard [74]. It is the task of the lowest layer of the TSS to abstract this driver and expose an OS and vendor independent set of functions that allows basic interaction with the TPM. This lowest layer is called the *Trusted Device Driver Library* (TDDL). The TDDL serves as a single-instance, single-threaded component and allows for sending commands as byte streams to the TPM and receiving the responses.

The next layer, the TSS Core Services (TCS), should be implemented as a singleton system service or daemon. It is the single instance that manages the

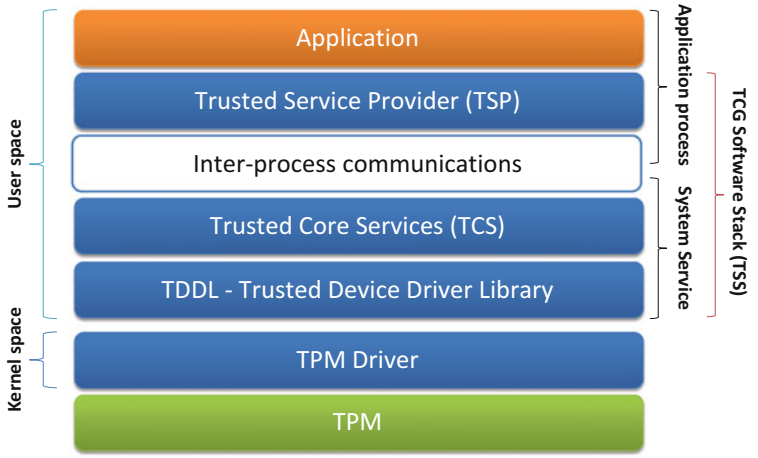


Fig. 1.1 The TCG software stack (TSS) architecture consists of several software layers within a trusted platform

TPM's resources and accesses it. It generates synchronized command streams from concurrent API commands to be transferred through the TDDL. The TCS takes care of the management of TPM key slots as well as permanent storage of TPM key material. Keys are assigned a Universally Unique Identifier (UUID) [34] that is used to identify stored keys. The TCS also maintains the SML where all PCR extend operations are recorded.

The upper layers of the software stack may access the TCS via inter-process communications according to the platform-independent Simple Object Access Protocol (SOAP) [78] interface.

The highest layer, the TSS Service Provider (TSP) provides Trusted Computing services to applications in the form a shared library. The TSP interface is defined as grouped function signatures and data structures in the C programming language. The TSS was also designed to allow partial integration with existing high-level APIs, such as PKCS #11 [52]. This enables the use of the cryptographic primitives provided by the TPM by legacy software. A limitation of this approach is that these legacy cryptographic APIs do not account for advanced Trusted Computing concepts such as sealing. Also the TCG's key typing and padding policies need to be considered [12] and might not match all application areas.

1.3.1.1 TSS in Embedded Systems

On Desktop systems, recent years have seen successful integration of generic TPM 1.2 hardware drivers into major operating systems, i.e. not only Windows, but also Unix derivatives and Linux. Several implementations of TSS exist. One noteworthy open source implementation by IBM is TrouSerS [25].

As we will discuss later in Sect. 1.6.2, there are different sizes and classes of embedded systems. Typically, however, only selected components of the TSS architecture are used. For instance on the TDDL level, drivers are usually derivatives of the Linux implementations. Which driver to use depends on the hardware interface of the TPM.

When drivers have established basic connectivity, higher layers may access the TPM. In many specific use cases, it is sufficient to implement only a small number of instructions, such as writing into a PCR register. This can be achieved by assembling the appropriate instruction byte arrays manually in programs [32] or by including command line tools. This keeps the software overhead on embedded platforms small; however, more complex instructions, for instance creating or certifying fresh keys are a tedious task to write, and most implementors will need a full-scale TSS to achieve such functionality. Very small platforms will note able to support this. One possibly solution can be to perform complex initialization and commissioning tasks on the TPM chip at manufacturing time, before even soldering it onto the target platform [79].

On medium-sized embedded systems, running a full TSS becomes feasibly, although the required XML and cryptographic features require a certain amount of system resources.

1.3.2 Review of Existing Java Libraries

This section presents an overview of existing libraries and APIs that provide first, experimental support for Trusted Computing to Java developers. Additionally, strengths and weaknesses of the individual approaches are discussed.

1.3.2.1 Trusted Computing for the Java Platform and jTSS

The central component of the open source "Trusted Computing for the Java Platform" project is an implementation of the Trusted Software Stack (TSS) for Java programs called jTSS [47]. It is a large library that provides Java programs with the TSS functionality that C programs currently enjoy.

Overall, the project offers two flavors of TSS implementations.

jTSS Wrapper Provides Java programs access to C-based stacks through an object-oriented API, which forwards calls to the native TSS. A thin C-back-end integrates the TSP system library. The Java Native Interface (JNI) maps the functions of the C based TSP into a Java front-end. There, several aspects of the underlying library, such as memory management, the conversion of error codes to exceptions and data-type abstractions, are handled. This wrapping approach results in complex component interactions. Unfortunately, debugging across language barriers is a challenging task. Another drawback is that implementation errors in the C-based components may seriously affect JVM stability.

jTSS Is a native implementation of the TCG Software Stack written completely in the Java language. It offers seamless support for Linux operating systems and all Windows versions starting with Vista, demonstrating platform independence. The Java TCS also synchronizes access from multiple Java applications. Such a full Java TSS implementation clearly reduces the number of involved components and dependencies. Consequently, this approach results in fewer side-effects from incompatible TSS implementations or different interpretations of the TSS specification. Moreover, a pure Java stack can easily be ported to other operating systems and platforms.

The API exposed by both variants is the same, enabling Java application programmers to switch between the two seamlessly, with the choice of the backend implementation depending on the surrounding platform. It defines data types, exceptions and abstract methods – we refer to it as the *jTSS API* [72]. It closely follows the original TSS C interface, permitting the user to stay close to the originally-intended command flows and providing the complete feature set of the underlying library. jTSS covers almost all of the functions specified by the TCG for communicating with the TPM at the fine granularity of TSS commands. As with every TSS, a complex sequence of commands is required to achieve functionality such as sealing, binding, attestation and key generation for application software. The project also provides `jTpmTools` (jTT), which is a collection of useful sample programs. They are implemented using jTSS API and demonstrate its usage.

The libraries were first created in the course of the Open_TC [42] research project. The first release was authored by Winkler, while the subsequent releases, maintenance and support activities have been done by Toegl. jTSS has since become a popular choice for Trusted Computing related research activities [2–5,7,8,11,13,15–17,22,24,27,29,31,33,37,38,44–46,55,59,63,66,67,69,70,81–84] and even found (quite) limited industrial use. It is one of the most widely used, supported and regularly updated TSSes available today.

While jTSS does allow the use of the TPM from Java, it still involves a significant learning curve for the average Java programmer, who may not be familiar with the procedural programming style that stems from the C-based TSS legacy. Overall, it is still a complicated API that requires a large amount of training and cross-language experience before it can be used in real-world projects.

1.3.2.2 TPM/J

Sarmenta et al. presented TPM/J [53,54], a high-level API that allows Java applications to communicate with the TPM. It is compatible with the Linux, Windows XP and Windows Vista and Mac OS X[1] operating systems thus living up to the promise of platform independence.

[1] At the time of writing, the inclusion of TPMs in Mac OS X compatible platforms has been discontinued.

Strictly speaking, TPM/J is not a TSS since it intentionally deviates from the specifications of TCG's TSS, which seems natural since the TSS specifications provide details specific to the structural programming paradigm and cannot be ported to the object-oriented perspective without major changes to the specifications. A drawback is that the library does not feature a split design as the TSS. Therefore, the JVM must run with elevated privileges to access the TPM hardware resource. Moreover, a major concern for users of TPM/J is that it is not regularly maintained, thus making it unsuitable for large-scale adoption in the community. It has however been used to study monotonic counters [53], and an attack on the TPM [1].

1.3.2.3 TPM4JAVA

TPM4JAVA [23] is a Java library that provides an easy-to-use API to Java programmers for communicating with the TPM. Its design is based on three levels of abstractions:

1. High-level: It provides developers with conveniently usable functionality to execute selected commands such as taking ownership, computing hashes and generate random numbers.
2. Low-level: This is a less user-friendly approach that allows programmers to execute any of the commands supported by the TPM.
3. Back-end: It is used internally for communicating with the TPM device driver library.

While the high-level API makes several functions easily accessible, some operations, such as performing a quote during attestation, require several lines of code and a low-level understanding of the actual functioning of the TPM. This makes 'high-level' a misnomer. The project has not been maintained for several years. Finally, TPM4JAVA shares the limitation of the other approaches of not adhering to the TCG's specifications.

1.3.3 Other Proposed Higher Level Interfaces

From a developer's point of view, the highly complex TSS design suffers from several drawbacks. It is challenging to develop applications with it, as even straightforward mechanisms of the TPM correspond to complicated instruction flows in the TSS API. Implementations of the various TSS layers themselves are often difficult to maintain and suffer from a high risk of introducing security-critical errors. A lot of functionality specified in the API is not relevant for many typical use cases of Trusted Computing. This is especially true for heterogeneous environments or embedded platforms. Based on these insights, individual proposals for higher level interfaces have recently been made for non-Java environments.

Stüble and Zaerin [60] propose a simplified trusted software stack (μTSS) for the C++ language. It mimics the TCG layered architecture (in the form of object-oriented oTDDL, oTCS and oTSP layers), with oTSP providing high level abstractions of selected functionality. It is noteworthy that oTCS offers access to all TPM instructions. Currently, μTSS does not separate user and system processes for device access and does not offer automatic TPM resource management to applications.

Also for C++, Cabbidu et al. [10] present the Trusted Platform Agent, a library that aggregates TSS functions into a higher-level API and also integrates other features missing in the language's standard library like cryptography and network communications. It therefore provides selected building blocks for trusted applications that can be applied with a low learning curve.

Reiter et al. [51] describe an alternative stack design that integrates in an open source cryptography API for Microsoft .NET.

Beneath the TCS layer of the TSS, the TPM Base Services (TBS) [39] in Windows Vista or later, virtualize the TPM for concurrent access and also offer a small set of management features to scripting languages.

1.3.4 Findings

While the aforementioned APIs all share the common goal of providing Trusted Computing functionality to Java developers, to date none of them has seen widespread adoption beyond research and academia.

One of the main reasons is that the interfaces exposed by the libraries often are difficult to learn and understand. This stems from several facts:

1. Trusted Computing by itself is a complex technology. The specifications defining the two major components – the TPM and the TSS [73, 75] – together consist of about 1,500 pages. The concepts are often not well presented for novice users and details have to be looked up in several different places. So it comes as little surprise that very few actual software products make use of the original C-based TSS to access the TPM. Indeed, a 2008 study [57] on the TSS concludes, that, "*it is apparent that, until now, no application exists that makes use of this technology. Even the simplest applications [. . .] have not been applied yet.*"
2. Implementations like jTSS try to mimic the interface defined by the TSS specification. This interface, however, was developed for procedural programming languages like C. Even though jTSS tries to map the TSS concepts to an object oriented API, it still does not fit well into the Java ecosystem and feels unnatural to developers familiar with other Java APIs. Furthermore, large and complex amounts of code are required to set up and perform basic Trusted Computing functions. This stems from the fact that in the original C-based TSS API, functions take long lists of parameters with many potentially illegal combinations. This makes the API error prone and complex to use for developers without detailed Trusted Computing knowledge.

3. Implementations like TPM/J and TPM4JAVA provide alternative interfaces to Trusted Computing functionality. While in the first case, the interface is at a very low level, the second one offers some higher level abstraction, but is neither consistent nor complete.
4. A full-fledged TSS is a very flexible and powerful library, but practical experience has shown that its full capabilities are not actually required for the vast majority of typical Trusted Computing applications. From our long experience of maintaining jTSS and supporting its users stems the insight that most adopters only follow existing code examples and test code. Few experiments create previously-not-demonstrated functionality, which requires a very steep learning curve.

Therefore, a novel design is needed that improves on the identified shortcomings and provides a programming interface suitable for Java developers while considering the specifics of trusted hardware platforms, legacy software architectures, and, obviously, the Java environment in different deployment scenarios, conventional or embedded.

1.4 API Design

We can now move on to describe the major influences on our specification and our resulting design decisions. Based on defined goals and clear assumptions on the developers that we target, we consider the restrictions imposed by the surrounding environment and discuss how the standardization process has influenced our proposal. From these constrains, we have implemented an agile specification process which enables us to derive the design of the JSR 321 API.

The Java Community Process (JCP) [28] aims to produce specifications using an inclusive, consensus-based approach. The specifications are, throughout their creation and also after release called "Java Specification Request" and assigned a running number. It is controlled by an elected *Executive Committee* (EC), which represents most major players in the Java industry. The central element of the process is to gather a group of industry experts who have a deep understanding of the technology in question and then have a technical lead work with that group to create a first draft. Consensus on the form and content of the draft is then built using an iterative review process that allows an ever-widening audience to review and comment on the document. While the JCP provides a formal framework with different phases and deliverables, an *Expert Group* (EG) may freely decide on its working style.

There are a number of phases in the process. At first, a new specification is *initiated* by a community member and approved for development by the EC of the JCP. Then, a group of experts is formed to develop a preliminary draft of the specification. Feedback from *early reviews* is used to revise and refine the draft. Once considered complete, the draft goes out again for *public review*. Now, the

EC decides if the draft should proceed. If approved by the EC, a proposed final draft of the specification is published and the leader of the expert group sees that the reference implementation and its associated technology compatibility kit are completed. Then the EC will decide on its *final approval*. Completed specifications will be *maintained* and updated.

The process also requires a *reference implementation* (RI). Its purpose is to show that the specified API can be implemented and is indeed viable. With the *Technology Compatibility Kit* (TCK) a suite of tests, tools, and documentation that is used to test implementations for compliance with the specification has to be provided as well.

1.4.1 Goals for a Novel API

From the previously outlined study of different existing Trusted Computing libraries we conclude that none of the proposals fulfills the desirable features an industry specification for applied usage also outside of the academic niche should have. We therefore propose a new API and present a set of goals the Expert Group has decided on.

Integration with Trusted Computing Platforms: As a software interface, the API should be oblivious to the actual hardware it is running on and should not introduce additional limitations on hardware resources. To the OS, the Java Virtual Machine appears just as an ordinary application. Therefore, the TPM access mechanisms need to integrate [69] with the surrounding environment of the OS, be it virtualized or not, and management services.

User-Centric Design. An Application Programming Interface is directed towards the programmer. An Trusted Computing API should therefore be designed to aid developers in writing security applications under the assumption of defined knowledge in the field of application.

Simplified Interface. To make the new API fit into the Java ecosystem, a completely new and fully object-oriented interface is to be designed. For instance, generic entities (e.g., cryptographic keys) in the TSS should be replaced with specific classes that represent the different types (e.g., a dedicated class for each type of key). This allows the set of offered operations to be limited to those actually applicable for a certain object type, thus enhancing usability and reducing the risk of errors.

Reduced Overhead. The TSS API requires a substantial amount of boilerplate code for routine tasks, such as key creation, data encryption or password management. The proposed API should attempt to replace these lengthy code fragments with simple calls using sensible default parameters where required.

Conceptual Consistency. Names in the API should be consistent not only within the API but also with the nomenclature used by the TCG and in Trusted Computing literature. This will allow users to easily switch from other environments to the proposed API. Still, naming conventions of Java must be adhered to.

Testable and Implementable Specifications. The API design should target a small core set of functionality, based on the essential use cases of Trusted Computing. This restriction in size will allow for complete implementations and functional testing thereof. Also, limitations of scope make it possible for all implementations to cover the full proposed API, a key requisite for true platform independence.

Extendibility. The API should allow implementers and vendors to add functionality which is optional or dependent on the capabilities of the surrounding platform.

Standards Compliance. Having an industry-wide standard of accessing the TPM from software is indispensable for widespread use and for enabling code mobility. As the TSS API has shown itself to be unfit for Java environments, the newly proposed API should itself be based on a novel, independent industry standard.

1.4.2 Expected Developer Knowledge

A major goal of the proposed JSR 321 API is to simplify Trusted Computing and make it accessible to a larger group of software developers. To achieve this, it is essential to understand our target audience and their skills before we can move on to create a programming interface for them. In the following we define which skills and knowledge we expect of a developer in order to make full use of the API.

In general, a developer using JSR 321 should be familiar with the cryptographic mechanisms provided in the Java Security Architecture [19, 36]. The concepts of data encryption, decryption and the creation of message digests using hash algorithms should be familiar. The algorithms in particular include SHA-1 and RSA as used by current TPM implementations. Moreover, a general understanding of Trusted Computing concepts and the functionality provided by a TPM is required. This at least should include knowledge about the following topics:

TPM Life-cycle. Starting with its manufacture, a TPM goes through a number of different states. A developer must understand this life-cycle, for instance that the TPM is shipped in an unowned state and its owner must explicitly take ownership, activate, and enable it. When the machine containing the TPM reaches its end of life, the TPM may be cleared to ensure that any TPM protected data can no longer be accessed. To avoid data loss, appropriate mechanisms like key backup or migration must be executed beforehand. Also the implications of a transfer of ownership of a platform need to be considered.

TPM Key Management. A TPM supports a range of different key types, including storage, binding and signature keys. The developer is responsible for building and maintaining a consistent hierarchy. For instance, if certain keys are created as non-migratable this may rule out any backup of them.

Root and Chain of Trust. Ideally a consistent chain of trust would be established by the operating system. However, today's mainstream platforms fail to do so. Developers need to take extra care to consider the security level represented by the PCR values.

Trusted Storage. Care must be taken when the binding and especially sealing mechanisms are applied to data or user supplied key material. Again, the problem of backup arises, especially considering state changes which can render sealed data permanently unaccessible.

Attestation. A number of different protocols have been proposed to perform attestation to a remote verifier [62]. This API supplies the means to create TPM quotes. Since there is no general standard for attestation and public key infrastructures remain limited, we leave the full specification and implementation of communication protocols to the application developer.

1.4.3 API Scope Considerations

JSR 321 aims to be a simplified, compact and user-friendly API, that should integrate in the complex ecosystem of today's Trusted Computing infrastructures. It it therefore imperative to clearly focus the functions offered by the interface so that the resulting design is consistent and viable.

A natural starting point would be to derive a Trusted Computing API from the complete TSS specifications. However, JSR 321 is not planned to fully replace the TSS in all its tasks. Instead, and as required by the nature of the JVM as a user process, it builds on and extends the TSS services offered by the operating system environment.

As a deliberate design decision, JSR 321 will provide functionality focused on applications and middle-ware, rather than providing support for the low level BIOS or OS features of the TPM. This restriction matches the field of use of Java and permits a significant reduction in complexity. JSR 321 will not duplicate elements of the Java Cryptography Architecture, thus fitting into the existing library framework.

Finally, many TSS-specified functions are simply not needed in Java APIs. Management of memory and other resources can and should be hidden from application developers. Object initialization and destruction are natural features of object-oriented languages. Cryptographic primitives like hash functions are already well-supported in the Java Cryptography Extension.

To derive the functional scope of the API, the commented complete list of TCG-specified TSP functions [12] was considered. Based on the criteria and principles laid out previously, those features were selected that are required for core use cases that have high importance for practical applications.

In summary, the design focuses on the most important *core concepts of Trusted Computing*. The second main goal is to provide *high usability*. At the same time, the API is designed to remain modular enough to be extensible with future developments.

1.5 Outline of the API

The unique name-space officially assigned to the JSR 321 API is `javax.trusted computing`. Within this name-space, a number of packages has been specified, each representing a well defined set of functionality in several classes. The relationship between the packages and classes is outlined in Fig. 1.2. These packages comprise the API:

`javax.trustedcomputing.tpm` This package contains all relevant functionality for connecting to a TPM. Within the TPM hardware chip, the concept of a *context* allows the separation of objects such as keys between different users and different connection sessions. In JSR 321, a TPM connection is represented by the central `TPMContext` object that acts as a factory for other objects specified by the API such as the `KeyManager` or the `Sealer`. The `TPM` interface is also defined in this package, which provides general TPM related information such as its version and manufacturer. Additionally, it allows PCR registers to be read and extended, as well as providing the `Quote` operation required for platform attestation.

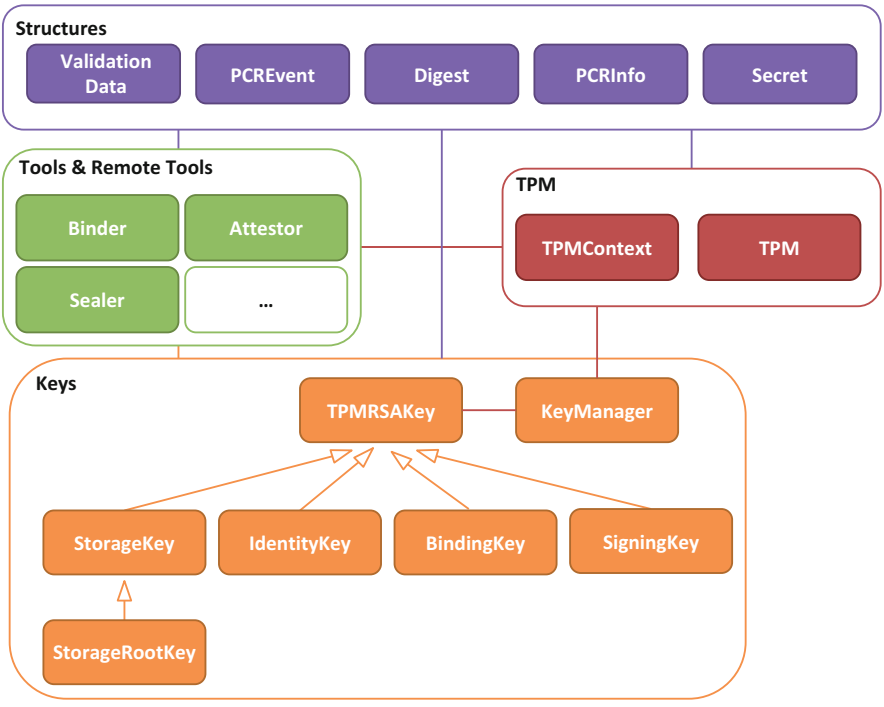


Fig. 1.2 Illustration of the relationship between the core components, including the TPMContext, KeyManager, and Key classes and the Tools

`javax.trustedcomputing.tpm.keys` Contrary to the TSS specification, JSR 321 introduces specific interfaces for the individual key types supported by the TPM. This includes interfaces for storage, sealing and binding keys. Compared to having one generic key object, this approach reduces ambiguities in the API and allows appropriate key usage to be enforced at the interface level. Using strong key types also relates well to results in formal API design and analysis research.

`javax.trustedcomputing.tpm.structures` This package holds data structures required for certain TPM operations. They include the `PCREvent` structure required for operations on the measurement log, `PCRInfo` used as part of platform attestation and `ValidationData` as returned by the TPM quote operation.

`javax.trustedcomputing.tpm.tools` In this package, there are interface definitions for helpers classes to perform TPM operations such as binding, sealing, signing and time stamping. The `javax.trustedcomputing.tpm.tools.remote` sub-package offers abstract classes that allow a remote host without TPM to participate in Trusted Computing protocols. It provides the functionality to validate and verify signatures on TC data types.

For error handling, a single `TrustedComputingException` covers all lower layers. It offers the original TPM/TSS error codes, but also a human readable text representation, which is a great step forward in terms of usability. Despite using only a single exception class, implementations of the API should forward as much error information as possible. For illegal inputs to the JSR 321 API, default Java run-time exceptions are used. Finally, functions offering bit-wise access to status and capability flags are replaced by specific boolean methods that allow access to application relevant flags.

In JSR 321, the `KeyManager` interface defines methods for creating new TPM keys. Upon creation, a secret for key usage and an optional secret for key migration have to be specified. After a key is created, the `KeyManager` allows the key, encrypted by its parent, to be stored in non-volatile storage. As required, the `KeyManager` allows keys to be reloaded into the TPM, provided that the key chain up to the storage root key has been established (i.e., each parent key is already loaded into the TPM). Every time a new key is created or loaded from permanent storage, a usage secret has to be provided. This secret is represented by an instance of a dedicated class `Secret` that is attached to the key object upon construction. `Secret` also encapsulates and handles details such as string encoding, which are often a source of incompatibility between different TPM-based applications.

The extendable `tools` package implements various core concepts of Trusted Computing. As each tool that accesses the TPM is already linked to a `TPMContext` at creation, there are few or no configuration settings required before using the tool. Each tool provides a small group of methods that offer closed functionality. For example, a `Binder` allows the caller to `bind` data under a `BindingKey` and a `Secret`, and returns the encrypted byte array. Usage complexity is minimal as no further parameters need to be configured and the call to `unbind` encrypted

```
1 try {
2
3     TPMContext context = TPMContext.getInstance();
4     context.connect(null);
5
6     KeyManager keyManager = context.getKeyManager();
7
8     StorageRootKey srk = keyManager
9             .loadStorageRootKey(Secret.WELL_KNOWN_SECRET);
10
11    Binder binder = context.getBinder();
12
13    Secret keyUsageSecret = context.getSecret("Passphrase for
          using the key.".toCharArray());
14
15    BindingKey bindingKey = keyManager.createBindingKey(srk,
16            keyUsageSecret, null, false, true, true, 2048, null);
17
18    byte[] plainData = new String("Data to be encrypted and bound.
          ").getBytes();
19
20    byte[] boundData = binder.bind(plainData,
21                    bindingKey.getPublicKey());
22
23    context.close();
24
25 } catch (Exception e) {
26    // Handle errors..
27 }
```

Fig. 1.3 Example of JSR 321 code that performs binding of data

data is completely symmetric. Besides the core set of tools (`Signer`, `Binder`, `Sealer`, `Attestor`, `Certifier`, `Signer`), implementers of JSR 321 may add further sets of functionality. An example might be the tool `Initializer` which manages TPM ownership, if the Java library is implemented on an OS without tools for doing so.

In Fig. 1.3 we list source code that demonstrates the API. The example shows Java code that first opens a TPM context session, creates a non-migratable cryptographic key with the following key policy: the key is a child of the Storage Root Key, its usage authenticated with keyUsageSecret; there is no migration secret set as the key is non-migratable; it's volatile, requires authentication, is a 2,048 bit RSA key and is not restricted to a PCR configuration. Finally, the program binds data to the platform where the code is executed. This example also allows us to evaluate the expressiveness and complexity of writing code. In [61], Stüble and Zaerin use the number of Lines of Code (LOC) of code examples as measure to compare different Trusted Computing APIs. They compare implementations of the identical binding use case. According to them, achieving the same functionality requires 146 LOC with TSS, 30 with jTSS and 18 using μTSS. The JSR 321 program we present takes only 15 LOC. Besides this obvious reduction of code size, the naming

conventions used throughout the API allow the effective use of code-completion mechanisms found in modern Integrated Development Environments (IDE) such as Eclipse. In many cases, the IDE will automatically suggest a suitable parameter for method calls, thus considerably speeding up the development of trusted computing applications.

1.6 Experience and Outlook

1.6.1 Third Party Implementation and Teaching Experience

Our API design has already been adopted by a third party, indicating the viability of the JSR 321 approach. To improve the security of smart power meters, the TECOM research project [64] has independently created a JSR 321 implementation based on the Early Draft version of the specification in order to satisfy their need for a high-level Trusted Computing API for Java-based embedded systems. Their implementation was built on top of the previously described ţ TSS in C++ and Java. The feedback [56] from this external implementation effort has been very positive and helpful. Aside from minor ambiguities in the specification and small feature requests, no major difficulties were reported. TECOM concluded that JSR 321 "provides most functionality that the majority of users would probably need" and that the single interface layer and low level of background knowledge required gave it an advantage over other APIs.

In summer 2010, JSR 321 was used for teaching the 5th European Trusted Infrastructure Summer School (ETISS), at Royal Holloway, University of London. In the 90-min "TPM Lab" we provided an introduction to the central component of Trusted Computing, the Trusted Platform Module (TPM). The lab explained TPM activation control, basic operations, and high-level programming of the TPM with JSR321. The concept of the chain-of-trust was explored in a practical sealing experiment. On the available HP desktop machines with Infineon TPMs we provided pre-configured USB-memory sticks running Ubuntu Linux and Eclipse as development environment. Although many of the participants had either no experience with TPMs or with Java, about two thirds of them were able to complete the implementation of an unsealing program within less than 1 h. This clearly underlines the low initial threshold for using the JSR 321 API. JSR 321 was also used in two practicals for courses taught at Graz University of Technology. In the third year bachelor-class "Security Aspects in Software Development" of 2011, students were given the choice of implementing the signature component of a CA service either through JavaCard or TPM interfaces. The groups that chose JSR 321 did not require more supervision or advise than those choosing the more conventional technology. In the master-level "Selected Topics IT Security 1" class, in the summer term 2012 students were, among others, given the task to demonstrate remote attestation of an Android environment. To this end, they employed a software emulated TPM and accessed it through JSR231. As the Android environment is source compatible with Java, this caused no additional complexity.

Thus, JSR 321 has proven to be fit for implementation by third parties not involved in the specification. Also, the API can be used in academic teaching and Embedded Systems much like other existing security technologies.

1.6.2 Application in Embedded Systems

In this section we study whether it is (performance-wise) feasible to use Java as a runtime environment for Trusted Computing related applications in Embedded Systems. This section briefly describes the used platforms and the corresponding software configurations.

When considering trusted embedded platforms we have to distinguish two classes of platforms: first, PC like platforms including many "industrial PC" motherboards, typically based on ×86 compatible processors. They have a PC-style Southbridge controller which exposes an LPC bus interface. On this class of trusted embedded platforms the TPM is connected to the system using the standard LPC bus interface. Trusted embedded platforms in this category can be treated exactly in the same manner as desktop PC platforms without any loss of generality. The following sections focus on a second class of trusted embedded platforms, which do not provide a standard LPC bus interface and thus have to resort to alternative methods for connecting to a TPM. We concentrate our discussion on TPMs connected via an I^2C bus. This bus is supported by virtually any embedded microprocessor or micro-controller of interest, either through dedicated hardware blocks or through software-emulation using general purpose input/output pins.

The inter-IC (I^2C) bus [41, 80] was introduced by Phillips Semiconductors (now NXP Semiconductors) in 1982 to provide simple bidirectional 2-wire bus for communication between micro-controllers and other integrated circuits (ICs). Data transfers are 8-bit oriented and can reach speeds of up to 100 kbit/s in standard mode or 400 kbit/s in fast mode. Higher transfer speeds up to 5 Mbit/s can be achieved given that certain hardware constraints are met.

Currently no approved publicly available TCG standard for TPMs with I^2C interface exists, although several vendors have recently started shipping I^2C-enabled TPMs. The TCG's Embedded Systems Working Group has started work on a, currently unreleased, draft for a I^2C TPM interface specification based on the existing PC-centric TPM TIS [74] specification. At the time of this writing no final standard has been publicly available yet.

1.6.2.1 Benchmark Platforms

We chose three different systems to evaluate the performance of Java-implemented Trusted Computing libraries. At the time of performing the analysis, available TPMs were designed for the PC platform only and thus used the *Low Pin Count* (LPC) bus as an interface. To enable Embedded Computing scenarios, we created our own

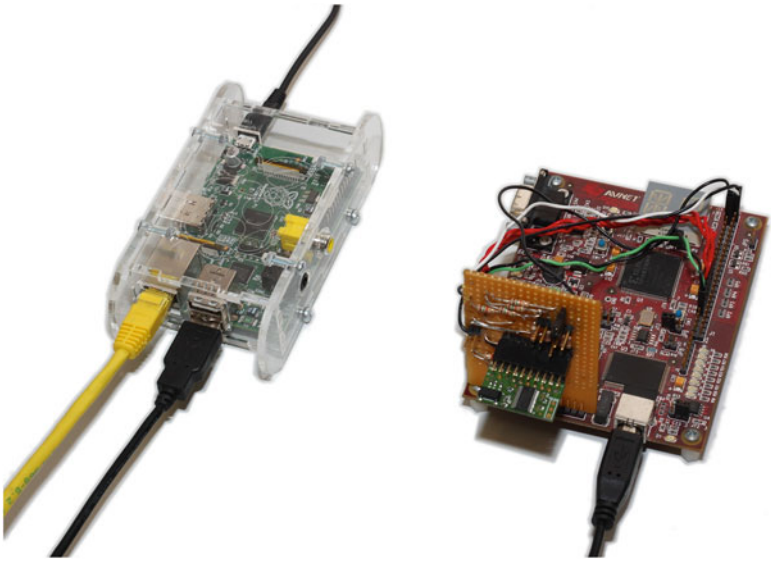


Fig. 1.4 Raspberry Pi with IAIK LPC-over-USB-TPM

adapter board which enables us to connect a LPC TPM to embedded systems. The board is described in more detail in [48] and can be seen in Fig. 1.4. Basically it tunnels LPC packets over an USB interface. In addition two kernel modules are built and loaded. They provide a `/dev/tpm0` device by accessing our LPC-over-USB adapter board. The used TPM is an Infineon TPM version 1.2 with firmware 3.17. This adapter in combination with the PC-TPM is used as substitute since TPMs with I^2C interface were not available at the time of the experiments.

To give a hint on how our adapter performs, a simple method was used. We measured the time the kernel module needs to obtain the content of all PCRs. The used command was `time cat /sys/class/misc/tpm0/device/pcrs`. On the HP dc7900 platform it takes 0.402 s with the built-in TPM and the `tpm_tis` kernel module. With the IAIK LPC-over-USB TPM and our kernel modules the command took 1.399 s. On a Raspberry Pi it finished after 1.520 s.

HP dc7900 We use an HP dc7900 desktop PC as a reference platform. It runs Ubuntu 12.04 x86_64 with OpenJDK 64-Bit Server VM,version 1.6.0_24, as provided by Ubuntu package repositories. The CPU is an Intel Pentium Dual-Core E5200. Together with 4 GB RAM this is not the newest generation of desktop PCs but provides a solid base for comparisons.

Freescale i.MX51 EVK The i.MX51 EVK is an evaluation kit for Freescale's i.MX51 system on a chip (*SoC*). The main component of the SoC is an 800 MHz ARM Cortex A8 CPU core. The platform has 512 MB of RAM, whereof about 400 MB are available for the operating system. The used OS is an Ubuntu 10.04

with a self compiled Linux kernel version 2.6.35. The sources and patches for the kernel are provided by Freescale. The Java environment is provided by several different Java Virtual Machines (JVMs):

- OpenJDK Zero VM, version 1.6.0_18, is the standard Java VM of Ubuntu 10.04. It comes without an JIT compiler, so the Java bytecode is just interpreted.
- Java HotSpot™ Embedded Client VM, version 1.6.0_32. This is a original binary release by Oracle, providing a Java SE 6 runtime environment for ARM 6 and 7 platforms. The JIT compiler is of the 'client' flavor.
- Java HotSpot™ Embedded Client VM, version 1.7.0_04, is an updated release based on the Java SE 7 specifications.
- Java HotSpot™ Embedded Server VM 1.7.0, version 1.7.0_04, offers a 'server' flavor (especially with regards to JIT compilation) VM for ARM platforms. In contrast to the 'client' VMs it is only available for ARMv7 processors.
- CACAO, version 1.6.0_25, this version was built from source to examine the performance of an open source VM with an included JIT compiler.

Raspberry Pi The Raspberry Pi is a very affordable computer targeting education and research. It is powered by an 700 MHz ARMv6 core in a Broadcom BCM283 SoC. On top of the SoC are 256 or 512 MB RAM[2] as package on package. Since the board is very popular and a broad user community has built up, there are many available operating system images available. For this survey the following three where used:

- Raspbian 2012-07-15
- Arch Linux 2012-06-13
- Debian 2012-06-18

While *Raspbian* is a Debian clone optimized for the Raspberry Pi, it is not binary compatible with Oracle's optimized JVMs. An overview of the available and considered JVMs follows here:

- OpenJDK Zero VM on Raspbian, version 1.6.0_24.
- OpenJDK Zero VM on Arch, version 1.6.0_22.
- OpenJDK Zero VM on Debian, version 1.6.0_24.
- Java HotSpot™ Embedded Client VM 1.6.0 This is the same JVM as on i.MX51 EVK.
- Java HotSpot™ Embedded Client VM 1.7.0 This is the same JVM as on i.MX51 EVK.

[2]The 256 MB version was used for benchmarking.

Table 1.1 Results for SciMark benchmark. Higher values are better. The `-large` command line option runs the test with larger matrices

Platform	JVM	Default ≈ *MFlops*	`-large` ≈ *MFlops*
dc7900	OpenJDK	845.45	500.32
i.MX51	OpenJDK	10.21	8.77
	Java6	30.34	21.02
	Java7client	30.05	21.13
	Java7server	31.35	24.78
	CACAO	24.83	17.53
Raspbian	OpenJDK	2.87	2.45
Arch Pi	OpenJDK	1.52	1.50
	Java6	21.66	14.41
	Java7client	21.23	14.13
Debian Pi	OpenJDK	2.86	2.43
	Java6	21.94	14.33
	Java7client	21.79	14.15

1.6.2.2 Benchmarks and Results

To initially assess the performance of generic Java software on the presented platforms, we used the SciMark benchmark suite. SciMark[49] is a rather simple benchmark examining the Java performance for scientific computations. It is provided by NIST and is chosen because its run time is not very long and it gives a brief insight in computational power of a Java environment. We use version 2.0a of the benchmark. The results for the SciMark benchmark are shown in Table 1.1.

The second benchmark was created to obtain performance data specifically relevant for Trusted Computing applications. As a base for the benchmark `jTpmTools` was used. The tasks performed in the benchmark are the following:

- `tpm_version`
  Query the TPM's version information
- `tpm_flags`
  Query the TPM's flag settings
- `pcr_read`
  Read the content of all PCRs
- `pcr_extend`
  Extend a small chunk of data to a PCR (20 byte)
- `pcr_extend_big`
  Extend a huge chunk of data to a PCR (40 MB)
- Sealing
  Seal and unseal 512 bit of zeros. This test consists of three tasks:
  - `create_key`

Create the needed key
  - `seal`
    Seal the data
  - `unseal`
    Unseal the data

- Quoting
  Obtain a quote from the TPM. This test consists of two tasks:
  - `create_key_legacy`
    Create the needed key
  - `quote`
    Obtain quote

- `stress_block_1` to `stress_block_10`
  Each block queries the TPM's flags 200 times, totaling in a count of 2,000 queries.

The resulting measurements are given in Table 1.2 and Fig. 1.5.

Table 1.2 Selected results of the Java-based tools benchmark. All values are *ms*, obtained with local bindings and file system storage. On dc7900 platform OpenJDK is used, on i.MX51 Java6 and on Raspberry Pi Java7client

Benchmark	dc7900	i.MX51	Raspberry Pi
tpm_version	143	161	158
tpm_flags	203	221	220
pcr_read	1,283	1,592	1,591
pcr_extend	278	284	555
pcr_extend_big	1,018	9,070	19,715
create_key	45,818	15,638	2,626
Seal	4,800	4,509	4,604
Unseal	3,239	2,389	2,418
create_key_legacy	18,170	15,135	2,243
Quote	6,000	5,829	6,000
stress_block_1	13,779	13,020	13,020
stress_block_2	13,511	13,019	13,058
stress_block_3	13,503	13,019	13,018
stress_block_4	13,547	13,019	13,028
stress_block_5	13,443	12,999	13,018
stress_block_6	13,695	13,019	13,048
stress_block_7	13,511	14,439	13,228
stress_block_8	13,635	14,999	13,028
stress_block_9	13,439	14,999	13,538
stress_block_10	13,587	14,999	13,898

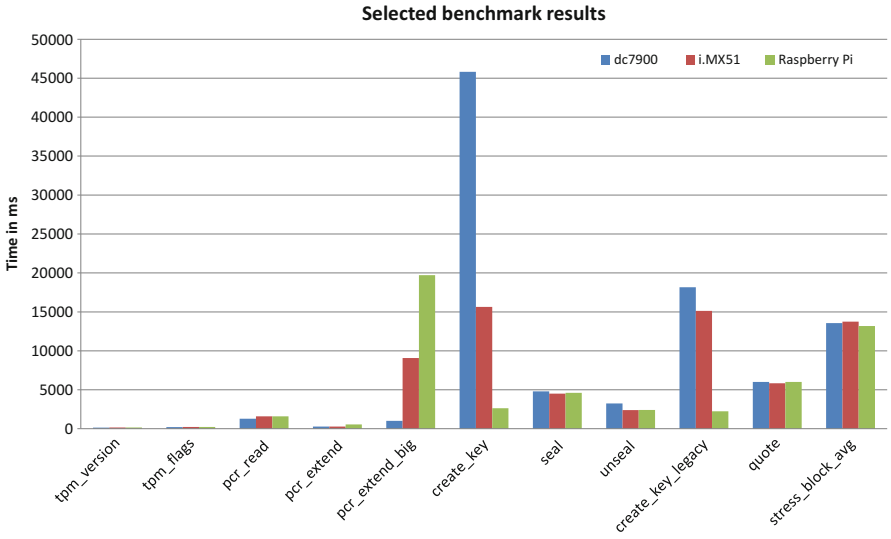


Fig. 1.5 Visualization of results presented in Table 1.2. `stress_block_avg` is the mean value of the 10 runs

1.6.2.3 Performance Discussion

The computational power of the investigated systems is by magnitudes lower than that of standard desktop PCs. Yet, the main insight is that it is possible to use a Java based trusted software stack also on limited platforms such as the i.MX51 EVK board and the Raspberry Pi. The results presented show that the performance difference between a desktop PC platform and embedded systems is minimal for Trusted Computing related tasks. This is due to the fact that TPMs are rather slow devices, so that even the medium-sized Embedded Systems we considered for our benchmarks need to wait for the results of the hardware security component.

Note that the values for `create_key` and `create_key_legacy` might be misleading. The creation of keys depends on entropy generated inside the TPM. The true random number generator's entropy pool is depleted after a few generated keys so any further creation has to wait for new randomness to be available. This leads to high deviations of the values. We show the minimum runtimes measured.

1.6.3 Compatibility with Next Generation TPMs

Throughout the standardization process, 1.2 was the latest version of the TPM specifications [75] officially released and therefore was used as basis for the design of JSR 321. We believe that our basic approach of providing a high-level abstraction of core concepts of Trusted Computing will remain valid for future versions of the

specification. Any necessary changes to the API, which could become opportune by the design of the TPM 2.0, which has recently been made available in a preliminary version [76], can be supported by releasing a new revision through the Maintenance phase of the Java Community Process. In addition, we believe that the results of the JSR 321 specification could itself serve to help guide the specification of the next generation of TSS. The authors have received encouraging feedback from the TCG that a high-level Trusted Computing API approach as pioneered in the presented work would be highly desirable for specifications in other languages such as C.

Summary

In this chapter we outlined the current status of software libraries for TPM access and application-level integration of Trusted Computing. To date, several commercial and some free implementations of the TCG Software Stack have been published with varying levels of completeness and standard compliance.

As a basis for the work presented in this chapter, we reviewed and discussed the state of the art of available Trusted Computing software libraries. For native applications, there are ongoing projects which aim to fully implement the TSS standard, and alternative approaches which intentionally provide a reduced and simplified interface.

In managed run-time environments, Java currently is the primary choice for the implementation of Trusted Computing applications. This is emphasized by the existence of several different libraries and frameworks that have been proposed or prototyped for this language. Our review of existing approaches has uncovered a number of drawbacks including high complexity, inconsistent APIs, limited object-orientation or lack of features.

Despite the availability of libraries and tools, Trusted Computing is not yet widely used and has not found its way into commercial applications [57]. We and other designers of TC APIs [60] attribute this fact primarily to the high complexity of and developer expertise required by existing standards and APIs. We believe that a lower learning curve for the software interfaces can attribute to a more widespread use in the future.

Based on these findings, we specify goals for a novel high-level Java API that aims to overcome these limitations. Specifically, we focus on a simple interface for access to commonly used TPM functionality and define the technical knowledge expected of programmers using it. In contrast to the original TSS design, we propose a fully object-oriented approach that hides low-level details and provides additional guidance for developers by providing solid default configurations. Results from the reference implementations are encouraging and demonstrate the feasibility of the proposed approach.

(continued)

Performance evaluations on Embedded Systems underline the applicability of our approach in different usage scenarios, that need not be restricted to the Desktop and Server world.

The aim of our API design was the release as the official Java standard API for Trusted Computing. Therefore, we have adopted an agile and transparent working style within the Java Community Process. The desirable set of API features has been selected based on open discussions. Feedback received from external reviewers and independent implementers has helped to adapt and extend the design. After two publicly announced reviews and several votes within the Java Community Process, the standard was published [68]. The Reference Implementation, the Technology Compatibility Kit and documentation is available under an open-source license from https://jsr321.java.net/.

We believe that this effort towards an open, simple and consistent programming interface can considerably contribute to the future adoption of Trusted Computing. Even thought the proposed JSR 321 API is designed for the Java programming language, we anticipate that the contribution of this work will not be limited to Java. Due to the clear and lightweight design of the API, implementations in other object-oriented programming languages should be possible with only minor adaptations.

References

1. Ables, K.: An alleged attack on key delegation in the trusted platform module. MSc Advanced Computer Science First Semester Mini-Project, University of Birmingham (2009). http://www.computer-science.birmingham.ac.uk/~mdr/research/papers/pdf/09-ables-3.pdf. Website accessed 15 Nov 2012
2. Alam, M., Zhang, X., Nauman, M., Ali, T.: Behavioral attestation for web services (ba4ws). In: Proceedings of the 2008 ACM Workshop on Secure Web Services, Alexandria, pp. 21–28. ACM (2008). doi:10.1145/1456492.1456496
3. Alsouri, S., Dagdelen, O., Katzenbeisser, S.: Group-based attestation: enhancing privacy and management in remote attestation. In: Acquisti, A., Smith, S., Sadeghi A.R. (eds.) Trust and Trustworthy Computing. Lecture Notes in Computer Science, vol. 6101, pp. 63–77. Springer, Berlin/Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-13869-0_5
4. Baldwin, A., Dalton, C., Shiu, S., Kostienko, K., Rajpoot, Q.: Providing secure services for a virtual infrastructure. SIGOPS Oper. Syst. Rev. **43**(1), 44–51 (2009). doi:10.1145/1496909.1496919
5. Bangerter, E., Djackov, M., Sadeghi, A.R.: A demonstrative ad hoc attestation system. In: Wu, T.C., Lei, C.L., Rijmen, V., Lee D.T. (eds.) Information Security. Lecture Notes in Computer Science, vol. 5222, pp. 17–30. Springer, Berlin/Heidelberg (2008). http://dx.doi.org/10.1007/978-3-540-85886-7_2
6. Bellare, M., Rogaway, P.: Optimal asymmetric encryption – how to encrypt with RSA. In: Santis A.D. (ed.) Eurocrypt 94 Proceedings, Perugia. Lecture Notes in Computer Science, vol. 950. Springer (1995). http://cseweb.ucsd.edu/~mihir/papers/oaep.html

7. Brett, A., Kuntze, N., Schmidt, A.: Trusted watermarks. In: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, 2009 (BMSB '09), Bilbao, pp. 1–7 (2009)
8. Brett, A., Leicher, A.: Ethemba trusted host environment mainly based on attestation (2009). http://ethemba.novalyst.de/wordpress/wp-content/uploads/2009/11/ethemba1.pdf. Website accessed 15 Nov 2012
9. Brickell, E., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, Washington, DC, pp. 132–145. ACM (2004). doi:http://doi.acm.org/10.1145/1030083.1030103
10. Cabiddu, G., Cesena, E., Sassu, R., Vernizzi, D., Ramunno, G., Lioy, A.: The trusted platform agent. IEEE Softw. **28**, 35–41 (2011). doi:http://doi.ieeecomputersociety.org/10.1109/MS.2010.160
11. Celesti, A., Salici, A., Villari, M., Puliafito, A.: A remote attestation approach for a secure virtual machine migration in federated cloud environments. In: 2011 First International Symposium on Network Cloud Computing and Applications (NCCA), Venice, pp. 99–106 (2011)
12. Challener, D., Yoder, K., Catherman, R., Safford, D., Doorn, L.V.: A Practical Guide to Trusted Computing, 1st edn. IBM Press, Upper Saddle River (2008). ISBN-13: 978-0132398428
13. Coppolino, L., Jäger, M., Kuntze, N., Rieke, R.: A trusted information agent for security information and event management. In: Proceedings of the Seventh International Conference on Systems, Saint Gilles (ICONS 2012). Think MInd (2012)
14. Dietrich, K.: Anonymous client authentication for transport layer security. In: De Decker, B., Schaumüller-Bichl I. (eds.) Communications and Multimedia Security. Lecture Notes in Computer Science, vol. 6109, pp. 268–280. Springer, Berlin/Heidelberg (2010). doi:10.1007/978-3-642-13241-4_24
15. Dietrich, K., Pirker, M., Vejda, T., Toegl, R., Winkler, T., Lipp, P.: A practical approach for establishing trust relationships between remote platforms using trusted computing. In: Barthe, G., Fournet, C. (eds.) Trustworthy Global Computing. Lecture Notes in Computer Science, vol. 4912, pp. 156–168. Springer, Berlin/New York (2008)
16. FABBRI, F.: Progetto e realizzazione di un protocollo di verifica dell'affidabilita' di un terminale remoto (In Italian). Tesi di laurea specialistica, Università di Pisa (2007)
17. Gissing, M., Toegl, R., Pirker, M.: Management of integrity-enforced virtual applications. In: Lee, C., Seigneur, J.M., Park, J.J., Wagner, R.R. (eds.) Secure and Trust Computing, Data Management, and Applications. Communications in Computer and Information Science, vol. 187, pp. 138–145. Springer, Berlin/Heidelberg (2011). http://dx.doi.org/10.1007/978-3-642-22365-5_17
18. Global Industry Analysts Inc.: Embedded Systems: Market Research Report. http://marketpublishers.com/ (2013)
19. Gong, L., Mueller, M., Prafullch, H.: Going beyond the sandbox: an overview of the new security architecture in the java development kit 1.2. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, pp. 103–112 (1997)
20. Google Inc.: Android OS. Available online at: http://www.android.com/ (2013)
21. Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A.: The Java Language Specification Java SE 7 Edition. JSR 901 (2011). http://docs.oracle.com/javase/specs/index.html. Website accessed 2 Nov 2012
22. Hein, D.M., Toegl, R., Kraxberger, S.: An autonomous attestation token to secure mobile agents in disaster response. Secur. Commun. Netw. **3**(5), 421–438 (2010). doi:10.1002/sec.196. http://dx.doi.org/10.1002/sec.196
23. Hermanowski, M., Tews, E.: Tpm4java. Currently only available through http://web.archive.org/web/20090510093615/http://tpm4java.datenzone.de/trac (2009). Website accessed 6 Nov 2012
24. Huh, J.H.: Trustworthy logging for virtual organisations. Ph.D. thesis, University of Oxford (2009)
25. IBM Corp.: Trousers – an open-source TCG software stack implementation. http://trousers.sourceforge.net/. Website accessed 30 Oct 2012

26. ISO: ISO/IEC 9899:2011 Information technology – Programming languages – C. International Organization for Standardization, Geneva (2011). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57853
27. Jang, J., Nepal, S., Zic, J.: A trust enhanced email application using trusted computing. In: Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing, 2009 (UIC-ATC '09), Maiden, pp. 502–507 (2009)
28. Java Community Process: JCP procedures overview. http://jcp.org/en/procedures/overview. For JSR 321, version 2.6 applied. Website accessed 12 Nov 2012
29. Jianhong, Y., Xinguang, P.: Protocol for dynamic component-property attestation in trusted computing. In: 2010 Second International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC), Wuhan, vol. 2, pp. 369–372 (2010)
30. Jonsson, J., Kaliski, B.: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational) (2003). http://www.ietf.org/rfc/rfc3447.txt
31. Khattak, Z., Sulaiman, S., Manan, J.: Security, trust and privacy (stp) framework for federated single sign-on environment. In: 2011 International Conference on Information Technology and Multimedia (ICIM), Kuala Lumpur, pp. 1–6 (2011)
32. Kinney, S.: Trusted Platform Module Basics: Using TPM in Embedded Systems, 1st edn. Newnes, Oxford (2006). ISBN 13:978-0-7506-7960-2
33. Korn, R., Kuntze, N., Repp, J.: Performance evaluation in trust enhanced decentralised content distribution networks. In: 2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), Naples, pp. 1–6 (2011)
34. Leach, P., Mealling, M., Salz, R.: A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122 (Proposed Standard) (2005). http://www.ietf.org/rfc/rfc4122.txt
35. Lindholm, T., Yellin, F., Bracha, G., Buckley, A.: The Java Virtual Machine Specification Java SE 7 Edition. JSR 924 (2011). http://docs.oracle.com/javase/specs/index.html. Website accessed 2 Nov 2012
36. Lipp, P., Farmer, J., Bratko, D., Platzer, W., Sterbenz, A.: Sicherheit und Kryptographie in Java (In German). Addison-Wesley, München/Boston (2000). ISBN 3827315670
37. Lyle, J.: Trustworthy services through attestation. Ph.D. thesis, University of Oxford (2009)
38. Lyle, J., Martin, A.: On the feasibility of remote attestation for web services. In: Proceedings of the 2009 International Conference on Computational Science and Engineering, Vancouver, vol. 03, pp. 283–288. IEEE Computer Society (2009). doi:10.1109/CSE.2009.213
39. Microsoft: TPM Base Services. Microsoft Developer Network. http://msdn.microsoft.com/en-us/library/aa446796(VS.85).aspx. Website accessed 30 Oct 2012.
40. Microsoft Developer Network: Overview of the .net framework. http://msdn.microsoft.com/en-us/library/zw4w595w.aspx. Website accessed 1 Nov 2012
41. NXP semiconductors: I2C-Bus Specification and User Manual (2012). Available online at: http://www.nxp.com/documents/user_manual/UM10204.pdf
42. Open_TC Consortium: The Open Trusted Computing Project (Open_TC) (2005–2009). Currently available only through http://web.archive.org/web/20110723233118/http://www.opentc.net/. Archived website accessed 30 Oct 2012.
43. Oracle: About Java (2012). http://www.java.com/en/about/. Website accessed 14 Nov 2012
44. Parno, B., Lorch, J., Douceur, J., Mickens, J., McCune, J.: Memoir: practical state continuity for protected modules. In: 2011 IEEE Symposium on Security and Privacy (SP), Berkeley, pp. 379–394 (2011)
45. Parno, B., McCune, J.M., Perrig, A.: Bootstrapping Trust in Modern Computers. Springer, New York (2011)
46. Pirker, M., Toegl, R., Hein, D., Danner, P.: A PrivacyCA for anonymity and trust. In: Chen, L., Mitchell, C.J., Martin, A. (eds.) Proceedings of the 2nd International Conference on Trusted Computing (TRUST 2009), Oxford. Lecture Notes in Computer Science, vol. 5471, pp. 101–119. Springer, Berlin/Heidelberg (2009)

47. Pirker, M., Toegl, R., Winkler, T., Vejda, T.: Trusted computing for the Java™platform (2009). http://trustedjava.sourceforge.net/. Website accessed 29 Jan 2013
48. Pirker, M., Winter, J., Toegl, R.: Lightweight distributed heterogeneous attested android clouds. In: Katzenbeisser, S., Weippl, E., Camp, L., Volkamer, M., Reiter, M., Zhang, X. (eds.) Trust and Trustworthy Computing. Lecture Notes in Computer Science, vol. 7344, pp. 122–141. Springer, Berlin/Heidelberg (2012). http://dx.doi.org/10.1007/978-3-642-30921-2_8.
49. Pozo, R., Miller, B.: SciMark 2.0(2000). http://math.nist.gov/scimark2/.
50. Ravi, S., Raghunathan, A., Kocher, P., Hattangady, S.: Security in embedded systems: design challenges. ACM Trans. Embed. Comput. Syst. **3**(3), 461–491 (2004). doi:10.1145/1015047.1015049
51. Reiter, A., Neubauer, G., Kapfenberger, M., Winter, J., Dietrich, K.: Seamless integration of trusted computing into standard cryptographic frameworks. In: Proceedings of the Second International Conference on Trusted Systems, Beijing, pp. 1–25. Springer (2011). doi:10.1007/978-3-642-25283-9_1
52. RSA Laboratories: PKCS #11 v2.20: Cryptographic Token Interface Standard. RSA Security Inc. Public-Key Cryptography Standards (PKCS) (2004). ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf Website accessed 29 Jan 2013
53. Sarmenta, L., van Dijk, M., O'Donnell, C., Rhodes, J., Devadas, S.: Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In: Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC '06), Alexandria, 1-59593-548-7, pp. 27–42. ACM (2006). doi:http://doi.acm.org/10.1145/1179474.1179485
54. Sarmenta, L., Rhodes, J., Müller, T.: TPM/J Java-based API for the trusted platform module (2007). http://projects.csail.mit.edu/tc/tpmj/. Website accessed 30 Oct 2012
55. Schlüter, M.: Realisierung einer mobilen, vertrauenswürdigen GeschÃd'ftsplattform auf Basis von Trusted Computing zur gesicherten Datenerfassung (In German). Master's thesis, Technischen Hochschule Mittelhessen (2012).
56. Schnepp, I., Panenka, S., Richard-Foy, M.: JSR321 feed-back from TECOM-FP7's implementation. Technical report, Atego (2010). Review 2.1
57. Selhorst, M., Stueble, C., Teerkorn, F.: TSS Study. Study on behalf of the german federal office for information security (BSI), Sirrix AG security technologies (2008). http://www.sirrix.com/media/downloads/57653.pdf,download. Website accessed 1 Nov 2012.
58. Shim, R., Mainelli, T., O'Donnell, B., Chute, C., Pulskamp, F., Rau, S.: Worldwide interfaces and technologies embedded in PCs 2010–2014 forecast. Technical report, IDC (2010)
59. Strasser, M., Stamer, H.: A software-based trusted platform module emulator. In: Lipp, P., Sadeghi, A.R., Koch, K.M. (eds.) Trusted Computing – Challenges and Applications. Lecture Notes in Computer Science, vol. 4968, pp. 33–47. Springer, Berlin/Heidelberg (2008). http://dx.doi.org/10.1007/978-3-540-68979-9_3
60. Stueble, C., Zaerin, A.: µTSS – a simplified trusted software stack. In: Proceedings of the 3rd International Conference on Trust and Trustworthy Computing (TRUST 2010), Berlin. Lecture Notes in Computer Science, vol. 6101. Springer (2010)
61. Stueble, C., Zaerin, A.: µTSS – a simplified trusted software stack. Technical report, Sirrix AG (2010)
62. Stumpf, F., Tafreschi, O., Röder, P., Eckert, C.: A robust integrity reporting protocol for remote attestation. In: Proceedings of the Second Workshop on Advances in Trusted Computing (WATC'06 Fall), Tokyo, Japan (2006). http://www.research.ibm.com/trl/projects/watc/FredericStumpfPaper.pdf
63. Tanveer, T., Alam, M., Nauman, M.: Scalable remote attestation with privacy protection. In: Chen, L., Yung, M. (eds.) Trusted Systems. Lecture Notes in Computer Science, vol. 6163, pp. 73–87. Springer, Berlin/Heidelberg (2010). http://dx.doi.org/10.1007/978-3-642-14597-1_5
64. TECOM Consortium: Trusted Embedded Computing project (TECOM) (2008–2010). Currently available only through http://web.archive.org/web/20100625044259/http://www.tecom-project.eu/. Website accessed 9 Nov 2012

65. Petazzoni, T. Opdenacker, M.: Java in embedded linux systems (2009). http://free-electrons.com/doc/embedded_linux_java.pdf
66. Toegl, R.: Tagging the turtle: local attestation for kiosk computing. In: Park, J.H., Chen, H.H., Atiquzzaman, M., Lee, C., Kim, T.H., Yeo, S.S. (eds.) Advances in Information Security and Assurance. Lecture Notes in Computer Science, vol. 5576, pp. 60–69. Springer, Berlin/Heidelberg (2009). doi:http://dx.doi.org/10.1007/978-3-642-02617-1_7
67. Toegl, R., Hutter, M.: An approach to introducing locality in remote attestation using near field communications. J. Supercomput. **55**(2), 207–227 (2011). doi:10.1007/s11227-010-0407-1. http://dx.doi.org/10.1007/s11227-010-0407-1
68. Toegl, R., Lipp, P., Nisewanger, J., Rao, D.D., Winkler, T., Keil, W., Hong, T., Nauman, M., Gungoren, B., Graf, K.M.: JSR321 Trusted Computing API for Java. Java Community Process Specification Final Release http://jcp.org/en/jsr/detail?id=321 (2011). Java Specification Request # 321. Website accessed 31 Oct 2012
69. Toegl, R., Pirker, M.: An ongoing game of tetris: integrating trusted computing in java, block-by-block. In: Gawrock, D., Reimer, H., Sadeghi, A.R., Vishik, C. (eds.) Future of Trust in Computing, pp. 60–67. Vieweg+Teubner, Wiesbaden (2009). http://dx.doi.org/10.1007/978-3-8348-9324-6_7
70. Toegl, R., Pirker, M., Gissing, M.: acTvSM: a dynamic virtualization platform for enforcement of application integrity. In: Chen, L., Yung, M. (eds.) Trusted Systems. Lecture Notes in Computer Science, vol. 6802, pp. 326–345. Springer, Berlin/Heidelberg (2011). http://dx.doi.org/10.1007/978-3-642-25283-9_22
71. Toegl, R., Winkler, T., Nauman, M., Hong, T.W.: Specification and standardization of a java trusted computing api. Softw. Pract. Exp. **42**(8), 945–965 (2012). http://dx.doi.org/10.1002/spe.1095
72. Toegl, R., Winkler, T., Pirker, M., Steurer, M., Stoegbuchner, R.: IAIK Java TCG Software Stack – jTSS API Tutorial (2011). http://trustedjava.sf.net. Website accessed 14 Nov 2012
73. Trusted Computing Group: TCG Software Stack (TSS) Specification Version 1.2 Level 1 Errata A (2007). http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification. Website accessed 29 Jan 2013
74. Trusted Computing Group: TCG PC Client Specific TPM Interface Specification (TIS) specification version 1.21 revision 1.00 (2011). http://www.trustedcomputinggroup.org/resources/pc_client_work_group_pc_client_specific_tpm_interface_specification_tis. URL http://www.trustedcomputinggroup.org. Website accessed 29 Jan 2013
75. Trusted Computing Group: TCG TPM specification version 1.2 revision 116 (2011). http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Website accessed 29 Jan 2013
76. Trusted Computing Group: Trusted Platform Module Library part 1: Architecture – Familiy "2.0" Level 00 Revision 00.96 (2013). http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Website accessed 1 July 2013
77. UBM Tech: 2013 embedded market study (2013). http://e.ubmelectronics.com/2013EmbeddedStudy/index.html
78. W3C XML Protocol Working Group: SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation, W3C (2007). http://www.w3.org/TR/soap12-part1/
79. Weiser, S., Tögl, R., Winter, J.: Measured firmware deployment for embedded microcontroller platforms. In: MeSeCCS Proceedings, Lisbon. SCITEPRESS (2014)
80. Winter, J., Dietrich, K.: A hijacker's guide to communication interfaces of the trusted platform module. Comput. Math. Appl. **65**(5), 748–761 (2013). http://www.sciencedirect.com/science/article/pii/S0898122112004634
81. Xingkui, W., Xinguang, P.: The trusted computing environment construction based on jtss. In: 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), Jilin, pp. 2252–2256 (2011)

82. Xinguang, P., Wei, J.: Filter-based trusted remote attestation for web services. In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Beijing, vol. 3, pp. 5–9 (2010). doi:10.1109/ICCSIT.2010.5564906
83. Yan, J., Peng, X.: Security strategy of DRM based on trusted computing. J. Comput. Inf. Syst. **9**(7), 3226–3234 (2011)
84. Zic, J., Nepal, S.: Implementing a portable trusted environment. In: Gawrock, D., Reimer, H., Sadeghi, A.R., Vishik, C. (eds.) Future of Trust in Computing, pp. 17–29. Vieweg+Teubner, Wiesbaden (2009). http://dx.doi.org/10.1007/978-3-8348-9324-6_2

Part II
Applications-Use Cases

Chapter 2
ARM® TrustZone®

Jon Geater

Abstract Millions of mobile devices are built around processors and systems-on-chip (SoCs) based on ARM® processor designs. The success of the ARM architecture is due in no small part to the fact that ARM only designs and licenses the base IP for SoCs: the company does not make or sell finished chips. Device makers in search of SoCs for their designs generally benefit from this model as they have a wide choice of products and vendors to choose from while enjoying the efficiency and reliability of standardized tool chains, low-level compatibility and a broad developer ecosystem. However in the area of security ARM-based devices were not always consistent or compatible, so ARM created TrustZone to provide a portable architecture-level security feature for the ARM community to build upon.

2.1 TrustZone Overview

TrustZone® is an architectural feature of the ARM® application processor architecture[1] that enables a single processor (or SoC) to run two quasi-independent software stacks, one so-called 'Normal World' (NWd) and one 'Secure World'[2] (SWd). The NWd runs the standard software stack that the user expects to see: Linux, Android or the like. The SWd runs a separate and (typically) unseen operating system that

[1] 'Application processor' as distinct from 'real-time processor' or 'micro-controller'. TrustZone as described in this text is only applicable on A class (recently referred to as Cortex-A) family processors, and not -R or -M class designs. Cortex-A is the type typically used in smartphones, tablets, smart TVs and the like, collectively referred to as 'smart mobile devices'.

[2] TrustZone nomenclature has seen a number of refinements over the years and the 'Secure World' label has generally given way to 'Trusted World' in contemporary documents, partly to differentiate from 'Secure Elements', and partly in recognition of the unattainable nature of full security and better reflecting the combinatorial and subjective role TrustZone plays in the wider computer system. Nonetheless architectural descriptions often retain the original 'Secure World' naming. This usage is reinforced by the naming of various architectural features that support TrustZone.

J. Geater (✉)
Trustonic Ltd., Cambridge, UK
e-mail: jon.geater@trustonic.com

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_2

provides security services to applications running in the NWd and to the NWd operating system itself. Although not technically enforced, by convention the SWd stack is very compact and contains only the minimum code necessary to service the security functions of the NWd software. The SWd therefore usually performs a function similar to a programmable security module; a slave to the NWd, and is often described in ARM materials as 'a virtual second core' or a 'secure virtual processor'.

Separation of the two software stacks is enforced in the processor hardware and communication between worlds is strictly moderated making a properly implemented TrustZone system highly resilient against software attacks even from a rogue[3] OS kernel. In addition TrustZone offers a highly integrated system security solution and is more than simple memory partitioning. Bus devices and memory regions can be shared by both words (either concurrently or switched) which is intended to enable powerful security designs while retaining a natural user experience and minimizing build costs [4].

With the knowledge that devices and memory can be shared it is important to understand that separation of the SWd and NWd is not symmetrical and in this special regard the virtual security processor analogy fails. As the lower security zone the NWd cannot access SWd resources, but the SWd can access all NWd resources, even those not explicitly identified for security functions. This behavior is logical given the architecture but still needs to be borne in mind when designing systems.

ARM TrustZone technology was introduced in 2002 and first made available to system builders in 2003 with the ARM 1176 processor [2].

2.2 Protection Target

TrustZone is intended to protect against software attacks. Various implementations of SoC designs incorporating TrustZone also include protection against simple hardware attacks (os 'shack attacks' [2]) but this quality of protection varies between designs. Protection against direct software-borne attacks is the central aim.

Over the past decade or so the rise of devices with application download capabilities, large data storage/manipulation capacity and high bandwidth internet connections has made mobile devices much more open to remote, scalable software attacks. While physical attacks remain a valid problem it is the likes of malware and other remote-origin threats that present the greatest opportunity for growth in the mobile device crime world.

[3]The notion of 'roguishness' is strictly subjective. For some, a user-built Linux kernel is the ultimate security breach, while for others it is the only acceptable environment for their usage. The design of TrustZone does not assume a particular semantic or detailed threat model in this regard: it is simply assumes that the entire NWd is untrustworthy from the perspective of the SWd.

To this end TrustZone attempts to solve that problem, preventing any software down to the level of supervisor mode code (usually the OS kernel, or a successful exploit of the same) from interfering with the device's most highly valued assets. In particular it aims to prevent the kind of mass, scalable attack that the Internet enables so well.

While theoretically usable for almost any situation, TrustZone has historically been used for a few common use cases. These are fairly simply split into two categories: system use cases, where TrustZone supports some underlying feature of the device and SWd operating system (e.g. firmware integrity, IMEI handling), and application use cases where TrustZone provides a security extension for (possibly downloaded) user software (e.g. Digital right management (DRM)). In all these cases, and others, the primary goal must be to protect individual instances from attack. Systems that rely on important shared class secrets are not a good fit for TrustZone since the security economics of physically breaking a few devices to compromise them all are quite attractive to the attacker, whereas the prospect of having to break (or at least touch) every device is much less appealing.

2.3 Architecture

TrustZone is an SoC-wide feature, meaning that it not only provides memory and process isolation but also enables trusted handling of AMBA 3 AXI bus[4] peripheral interrupts.

To achieve correct separation of SWd and NWd requires designed-in cooperation between the processor, memory and bus systems of the SoC. The processor implements the SWd and NWd as explicit operating modes ('secure' and 'non-secure' respectively), and memory and bus access requests then indicate which mode they belong to. This indication is given by the setting of the *NS bit*.

2.3.1 The NS Bit

The NS (or 'Non-Secure') bit is the central manifestation of TrustZone in the ARM processor architecture. It is a control signal that accompanies all read and write transactions to system bus masters, including memory devices. As the name suggests, the NS bit must be set low in order to access SWd resources.

[4] AHB and APB devices are not directly compatible with TrustZone – the signals are only properly preserved in AXI systems.

The NS bit is sometimes thought of as an extra address bit[5] that effectively partitions the memory space into two parallel logical regions: 32-bit space plus NS. This analogy makes TrustZone isolation and error behavior intuitive: attempts from NWd to access SWd memory will fail, even if it knows the exact 32-bit address, because the 33rd bit is different and so does not map to the desired memory location. The use of a control signal in this way makes for a very simple and elegant partitioning model since no special access control logic (which may be buggy, and would require its own execution context) is required.

Integrity of the NS bit state is vital to the correct separation of SWd from NWd so code making resource requests cannot set NS in the transaction directly: instead it is set, maintained, and checked by processor registers and bus components. Bus masters that are always deemed insecure should set NS = 1 in hardware to eliminate the chance of later problems. All other masters have their state configured by trusted firmware during system boot before any user code is able to execute.

2.3.2 The Monitor, World Switching and CP-15

Along with NWd and SWd, TrustZone introduced a third special execution context called Monitor mode. Monitor mode executes a very small piece of software – the Monitor – whose sole job[6] is to catch transactions, exceptions and interrupts that need to change between worlds and then handle that change robustly. The Monitor code is part of the SoC firmware[7] and may strictly do anything but in general it should only perform the minimum operations necessary to safely switch between NWd and SWd states. Given its ability to manipulate secure state and intercept secure interrupts the code of the Monitor needs to be small and robust, and ARM provides a sample to show the basics. Errors in the monitor can completely undermine world separation.

When NWd software wishes to contact SWd it must issue a Secure Monitor Call (SMC) instruction. This causes the processor to invoke the monitor which sets the state of the NS bit in the Secure Configuration Register in the System Control Processor (CP15) and banks sensitive registers to keep the system secure and consistent.

[5]The '33rd address bit' analogy (all TrustZone-supporting ARM systems were 32-bit when the feature was introduced: the implication for 40- and 64-bit systems is obvious) is not uncontroversial among those with intimate knowledge of processor and memory architectures, but it is close enough as to be useful.

[6]Technically any code can run in any mode, but this *should* be the sole job of the Monitor. Adding additional complexity to this powerful software can be extremely risky.

[7]The scope of 'firmware' is, of course, somewhat subjective. In practice the monitor is often provided by the authors of the rest of the privileged SWd code.

SMC calls are very simple: their single parameter is a 4-byte immediate value that indicates to the software in the SWd what service is being requested. This value is not interpreted by the processor in any way: it is simply passed through the monitor call. It is therefore up to the design of the SWd software and its NWd callers/drivers to agree conventional numbering and meanings for these values.

Although the analogy of the virtual security processor is appealing it must now be extended if it is to remain useful. Specifically all transactions with the security processor must be considered to be synchronous blocking transactions. This is clear because there is only really one processor[8] and it can only be in one mode at once, so when SWd tasks are running the NWd stops. While tasks can be swapped and interleaved this is not as simple as normal scheduled multi-tasking since it requires a trip through the Monitor. System designers must pay careful attention to this fact as spending large amounts of time in the SWd context can have serious adverse effects on device usability. This is another strong reason for the recommended model of SWd as a small-as-possible coprocessor and not a full stack.

2.3.3 Interrupt Handling

Interrupt handling is special in TrustZone-aware systems. The core of the processor actually implements three complete separate exception vector tables: one each for normal, secure and monitor contexts. This enables powerful and flexible routing of interrupts to the correct security context without the risk of the wrong code trapping a vital signal.

By convention TrustZone systems leave IRQ as an insecure (or 'normal', following the naming scheme) interrupt source and use FIQ as the secure interrupt source. This convention is not strictly required: it was simply a choice and recommendation made by ARM to be the least disruptive to existing software [2]. Both are initially trapped by the monitor which ensures the correct state is selected and restored before letting the interrupt be consumed.

Although the FIQ-as-secure-interrupt convention is not absolutely required it does have some architectural features behind it. CP15 includes a configuration register that prevents NWd software from modifying the FIQ mask bit but provides no such facility for protecting IRQs. This makes FIQ a much more reliable choice for implementing sensitive secure sources such as a watchdog timer.

[8] TrustZone was invented in an era when ARM SoCs were generally single processor, single core, and the design rationale reflects this. Modern SoCs have many cores, each of which has its own NWd and SWd, and many different configurations and control designs are possible. However since the complexity of shared resources, bus masters and power management control become much more complex in this scenario only the simple single processor case is addressed in this text.

2.3.4 Fabric Support

In order to successfully propagate the NS signal throughout the system there is a minimum amount of support required in the fabric of the SoC in addition to the basic processor. While system builders are able to add TrustZone support to almost any component they desire, this minimum set must be implemented else security integrity will be lost. ARM provides ready-built IP designs for each of these components but it is possible for the chip designer to use their own parts so long as the features are properly and coherently implemented.

As ever this list is incomplete for a variety of reasons, not least that chip designs vary so widely and their fabric requirements with them (a functioning MMU is required but absent from the list, for example). Three broad categories of fabric device cover the bulk of the less obvious essentials.

2.3.4.1 Cache Controller

If the cache controller were not TrustZone-aware it would be effectively impossible to implement shared/dual use memory areas: clearing the cache each time a call is made would introduce a significant performance burden, and failing to clear the cache would introduce severe risks of information leakage.

By having a cache controller that is aware of TrustZone it is possible to extend the "33rd bit" model to cache look-ups as well, which continues to afford systems the appearance of maintaining two separate address spaces for SWd and NWd. It also provides intuitive error behavior when invalid requests are made.

2.3.4.2 Generic Interrupt Controller (GIC)

In order to handle and route secure interrupts the interrupt controller needs to be able to directly choose which world (and so ultimately which OS and driver stack) receives which interrupts. By combining a security-aware GIC with monitor code it is possible to prevent NWd from interfering with sensitive SWd interrupt sources (secure watchdog, for example) while at the same time minimizing latency for handling other sources. Without this security awareness in the interrupt controlled the system would be forced into serious compromises on security and/or performance.

2.3.4.3 Address Space Controller

While it may be desirable to try to implement all of SWd in secure on-SoC memory,[9] it is often the case that the system will not have enough on-SoC dedicated memory

[9]This is sometimes achieved, and has a number of benefits: additional tamper resistance/evidence on one hand, and additional isolation potential on another.

to run the SWd OS and all its applications. An address space controller that is TrustZone aware enables the partitioning of the large system DRAM into general access regions (marking all transactions NS=1) and secure-only access regions (NS=0, just like inside the processor core). Memory locations in secure regions are only accessible by SWd code, meaning they are as strongly protected from software attack as the on-chip components.[10]

2.4 Pitfalls

TrustZone is a very powerful feature and care must be taken when implementing systems not to accidentally compromise security. As with all things there are almost infinite opportunities for error but a few common errors catch most basic problems.

2.4.1 Leaving Debug Features Enabled

ARM defines separate debug enable software signals for the NWd and SWd so that they can be configured and tested separately and safely. Leaving debugging features enabled and accessible in released designs (for example through JTAG, or a maintenance mode boot path without proper protections) potentially leaves the door open for attackers to take over the system.

2.4.2 Incorrect Management of the Memory System

Especially in the case of systems that use an address space controller to create secure regions of system DRAM it is important to ensure that the contents and access properties of potentially shared physical memory locations are properly managed. With dynamic memory systems much care must be taken to avoid secure memory from becoming readable or insecure information finding its way into the secure working memory.

It is also extremely important to ensure that when calls from the NWd to the SWd include memory addresses (e.g. for the shared command buffer) that these memory addresses be properly validated by the SWd code before they are used (either for reading or writing). SWd code has the potential ability to modify any system memory (including non-secure) and care must be taken to avoid SWd software being used to "do the dirty work" of a malicious NWd process.

[10]Off-chip components are of course more open to invasive hardware attacks such as probing.

2.4.3 Poor Handling of Firmware or Software Verification

With the possible exception of a small boot ROM, to a first approximation all software executing on a contemporary mobile device is first loaded from untrustworthy locations.[11] It is therefore necessary to implement strict integrity checking of all software that loads before essential configuration of secure peripherals and memory has been completed.

The security of the SWd software (and NWd software that relies on a particular SWd companion) depends entirely on the software that creates the execution environment (i.e. the NS and ASC configurations) and validates the integrity of the main SWd code (as above). Some known failures include:

- Race condition: checking the validity of one copy of an image but running another
- Incorrectly giving access to a debug mode (e.g. maintenance bootloader)
- False positive verification of bad signatures

2.4.4 Poorly Designed Application Interfaces

Even when the system design is perfect it is still very possible for application level errors to compromise security of the finished device. Consider the common example of a cryptographic token implemented in TrustZone: if the developers of that token create an API that allows keys to be taken out of the SWd and used in NWd software (rather than providing proxied use of the key in SWd) then the cryptographic token would not be providing very much security even when TrustZone is functioning flawlessly.

2.4.5 Insecure Use of Shared Buffers

A common pattern for message passing in TrustZone is to have a shared memory region accessible to both SWd and NWd. Even when the memory protections are properly set up at the low level, higher layers of software can make mistakes. If SWd code somehow works on this data in-place then its operation can be compromised by rogue (or even non-rogue) NWd software in all three fundamental ways:

- Confidentiality is lost in the case that SWd produces intermediate values that are not supposed to be seen by the caller

[11] This may refer to the obvious popular example of programs downloaded from the Internet but it is not intended to. At the level of TrustZone it is important to realize that system flash is untrustworthy since it can be modified by NWd.

- Integrity is lost in the case that NWd modifies the input data while it is being worked on (introduces TOCTTOU[12] errors)
- Availability is lost in the case that the NWd overwrites or deletes the information before the SWd has finished with it

2.4.6 *Incorrectly Configured Bus Peripherals and Bad Drivers*

Bus masters marked as secure permit their slave devices and thus drivers access to the SWd. Since driver code often performs powerful actions such as modifying page table entries[13] the potential to introduce errors here is large.

2.5 Standardized Software Environment

Being a designer of processor and fabric IP, but not of SoCs themselves, ARM has to define features at an architectural level rather than concrete implementations. In this context the NS bit is an extremely elegant solution but it fails to meet one of the major goals it set out to achieve: interoperability [3].

From the application writer's point of view TrustZone does not provide interoperability between implementations because it only provides the ability to create and enter secure state. No higher level functionality or APIs are defined that could actually satisfy any use case. To address this the SWd needs to have the option of a standardized software stack that performs known functions in a known way.

2.5.1 *TrustZone Software*

Around 2004 ARM and Trusted Logic S.A. together produced the first version of TrustZone Software that provided various low level platform security features such as device identification, secure storage, data integrity verification and I/O access control [5]. This provided application developers with a set of common APIs upon which to base broader secure systems.

This software was developed for a number of years until TrustZone API v3 was retired in 2011. After this date the chosen standard for interoperable software on

[12]Time Of Check To Time Of Use – a common security error whereby input data is correctly validated when presented, but is changed before being operated on. Famously responsible for a number of security errors in unix file handling.

[13]Since the architectural concept of a driver depends on the operating stack it is difficult to go into too many details of the software level problems that might arise. However low-level access to the memory system is a very portable and relevant concept.

TrustZone devices became TEE. It is important to stress that device makers are not required to adopt the standardized TEE today any more than they were required to run the TrustZone Software before. Custom software in SWd is completely possible but is clearly not interoperable with other vendors.

2.5.2 TEE

In keeping with the other concepts in this chapter, the Trusted Execution Environment (TEE) concept grows from the fact that SoCs are, by their nature, extremely general purpose devices. Defining software interfaces for very specific security use cases is therefore challenging and unlikely to be useful. Instead, a simple platform model was imagined which creates a programmable space where any companion code for any NWd use case can execute, free of threats from bad software in the large open operating system on the other side. The application author typically designs their application in two parts (code that manipulates sensitive data to go in the TEE, and code that does not to go in the open OS) and writes each part separately with a tightly coupled and consistent design. It is hoped, and often practically required (although not theoretically necessary) that the secure part of the code be much smaller than the other part. This has many benefits but the most obvious in security terms is that it vastly reduces attack surface and opportunities for exploitable bugs.

In its generic form the Trusted Execution Environment is not necessarily only for TrustZone: any number of physical manifestations make sense for a logically isolated operating environment. However in TrustZone systems the TEE is generically constructed from three major parts:

- The SWd software stack: simple operating system, secure devices, interrupt configuration etc.
- An internal API against which application writers write their sensitive code (enabling portability, simplicity and hiding the details of the underlying OS)
- A client API that enables communication between NWd programs and their SWd counterparts (hiding the complexities of SMC calls and the like, and usually taking care of shared buffers)

A leading interoperability standard for TEE, supporting both TrustZone and non-TrustZone implementations, is defined by the GlobalPlatform standards organization [4].

2.5.3 Role in Secure Boot

The SWd software stack enjoys a somewhat circular relationship with the system's secure boot process. At the most base level TrustZone and the TrustZone aware components make the SoC boot secure for SWd software by ensuring the device

resets into secure state and runs SWd before NWd, but of course this alone is not enough to protect a complex hardware/software system. Given the opportunity to run first by the hardware, the low-level software of the SWd stack must do its part to configure the platform properly and safely, particularly in configuring the address space controller and secure peripherals [1].

Summary
With great power comes great responsibility. If care is not taken in the SWd software then device security characteristics can be uncertain, but with the right accompanying software design ARM® TrustZone® is an elegant and powerful feature of Cortex-A processors that can provide significant protection of connected devices and applications against software-borne attacks.

References

1. ARM: ARM cortex A9 technical reference manual: system boot sequence (2004). http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/Chdfjdgi.html
2. ARM: ARM trustzone security whitepaper (2005). http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
3. Smackall Games: Enhanced security for mobile devices: A Q & A with ARM's Tiago Alves (2009). http://www.smackall.com/viewarticle.php?title=Enhanced-security-for-mobile-devices--Q-A-with-ARM-s-Tiago-Alves&article=816
4. GlobalPlatform: Truted execution environment (TEE) guide (2012). http://www.globalplatform.org/mediaguidetee.asp
5. IQ Magazine: TrustZone: integrated hardware and software security (2004). http://www.iqmagazineonline.com/magazine/pdf/v_3_4_pdf/Pg18_24_custZone_Secur.pdf

Chapter 3
Computer Security Anchors in Smart Grids: The Smart Metering Scenario and Challenges

Alessandro Barenghi, Luca Breveglieri, Mariagrazia Fugini, and Gerardo Pelosi

Abstract The modern energy distribution grid is increasingly responsible for extensive self-monitoring and load balancing, together with prompt failure response and small energy producer integration. In this scenario, where the grid doubles as a data transmission grid, commonly named smart grid, new information security challenges arise. In particular, providing confidential data transmission, privacy preserving metering and authentication for the metering software, emerge as key issues. In this chapter we will provide an overview of the security challenges of the smart metering scenario, highlighting both the security services to be guaranteed and their actors, and the ongoing standardization activities.

3.1 Introduction

In the latest years, social and policy changes have been driving the need for a reconfiguration of the current power grid, leading towards the integration of information and communication technologies (ICT) within the traditional energy distribution systems. The transformation of the power grid from a mostly unidirectional, centralized and hierarchical organization, to a distributed, networked and automated energy value chain, in turn spurred the need to integrate its management across a broad spectrum of heterogeneous business and operation domains, involving multiple enterprises and customers coming from different industries.

Smart meters, sensors, and analytics tools enable users to manage and control the energy usage in individual networked appliances as well as to automatically monitor the health state of the power grid, pinpoint outages as quickly as possible, and remotely assess the entity of eventual damages to grid assets, thus providing the grounds to locate and isolate the failure while maintenance teams are dispatched. Moreover, the integration of ICT enables energy and utility companies to better

A. Barenghi (✉) • L. Breveglieri • M. Fugini • G. Pelosi
Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Piazza L. da Vinci, 32, 20133, Milan, Italy
e-mail: alessandro.barenghi@polimi.it; luca.breveglieri@polimi.it; mariagrazia.fugini@polimi.it; gerardo.pelosi@polimi.it

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_3

evaluate the entity of power demand, possibly in near real-time, so that the delivery and independent production integration strategies may yield higher efficiencies altogether [12]. Beyond the application of traditional information-technology (IT) security mechanisms (such as authentication, secure protocols, and intrusion detection/response systems) together with proper security engineering processes, cyber-security in the smart-grid also faces novel challenges. In fact, the IT security approaches have to be reconciled with the traditional plant safety methodologies found in industrial control systems. This requires guaranteeing the stability of control systems, which may be disturbed by malicious activities. At the same time, IT security must take into account the real-time and analog nature of the grid and adapt the risk management by providing graceful degradation as opposed to a sudden, disastrous failure when under attack. The interconnection with other systems such as buildings and home networks also poses significant challenges in terms of consumer trust and the utilities' ability to manage encryption keys as well as the compliance with authorization policies satisfying the requirements of involved parties.

3.2 The Smart Metering Scenario

Smart meters are among the core components supposed to help transforming the energy delivery network into a two-way information system. They automatically measure the electric energy consumption of any end-consumer system (e.g., a single house, an enterprise, or even a whole urban block), and transmit the collected data to the utility provider, which in turn will automatically bill the consumer. They also take care of measuring a few technical parameters for the provider to balance the load in the power grid, and they disconnect and reconnect the consumer for either contract-related reasons (i.e., unfulfilled bill payments) or technical safety reasons (such as large power surges on the grid). Moreover, the same information can be used to manage emergencies and cope with grid faults, as well as to generate reports and statistics. Finally, as the main component of a two-way information system, a smart meter can notify pricing changes to final customers; they in turn will be able to automate the use of that heavy-load appliances such as cars and dishwashers work when the energy is cheapest. The automatic management of the utilities can be extended by means of smart metering and accounting, also to other commodity goods such as gas and water, through employing intelligent meters provided with a convenient data line to transmit the collected information.

Among different data connections, the one most suited for smart meters is performed via power line communications. A power line connection distributes electric power and data together, using the existing electric power grid to this purpose. It is characterized by having a reliability higher than that of the recent wireless communication technologies like the WiMAX [13] and GPRS/UMTS protocols, as well as by providing capillary access to all the households in a manner very similar to the common public switched telephone network or the local area network technologies.

Typically, a group of power meters, acting as data gateways, are connected to a concentrator, which can be conveniently placed within a mid to low voltage step-down substation. A low data rate, and an energy efficient wireless connection, most likely based on ZigBee [28], can be used to provide connectivity locally among nearby meters (e.g., those placed in a home or a building), which do not have a direct power line connection, such as the ones measuring non-electric goods like gas and water. For all such meters, the electric power meter works as a gateway to the power line connection.

At an intermediate level, the concentrators are connected to both the power grid and to an IP-based network (e.g., the global internet) in order to act as a data bridge to and from power lines. The electric power meter is able to communicate with the concentrator via the power line connection, and eventually the concentrator can communicate with the provider via a data network. Groups of distributors and providers can work in a Virtual Organization mode, e.g., they may share services and customers.

3.2.1 Architectural Reference: Actors and Services

Smart Metering involves four different actors: provider, distributor, consumer and meter. The first three roles, shown in Fig. 3.1, are identified through the following labels [8]: Energy Service Provider (ESP), Distribution Service Operator (DSO), and Final Customer (FC).

The ESP generates electricity from renewable and non-renewable energy sources in bulk quantities, and supplies it to the power distribution network. These sources are usually classified as: *renewable, variable sources*, such as solar and wind; or

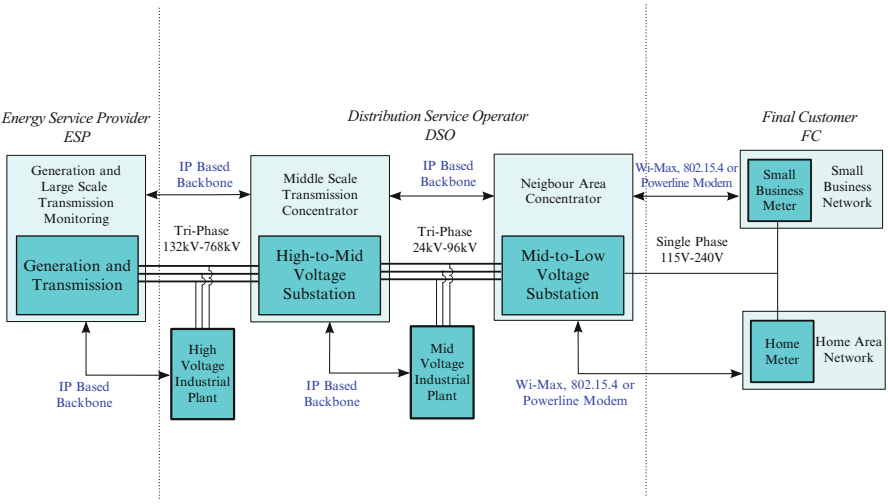


Fig. 3.1 Layered architecture of actors, components and communication in the power grid [1]

renewable, non-variable, such as hydroelectric, biomass, geothermal and pump storage; or non-renewable, non-variable, such as nuclear, coal and gas. The DSO, which may be the same entity as the ESP, manages (acquires and uses) metering data from the power grid. Metering data can be of two types: *consumption data*, namely the consumption of a resource (e.g., electric power) used for billing, and *technical data*, namely technical information used for power grid management, with the purpose of balancing the energy levels, of avoiding or smoothing load peaks, and of disconnecting and reconnecting customers and providers. The DSO uses consumption data for payments/billing, and for formulating custom billing plans and contracts with the "fidelized" FC (private or enterprise). It monitors and controls the status of the access points and bridges, or concentrators, through the acquisition of technical data about the meter statuses (e.g., enabled, disabled or faulty). It also manages emergencies, generates reports on consumptions based on consumption data about groups of consumers, and, in general, it performs measurements. It controls the functionalities of the meters to check if they are operative, and it is also able to send maintenance commands like "change billing contract", "connect" and "disconnect". Consumption data may also be used to generate reports and statistics, so generating consumption profiles (e.g., for certain user classes). Since a DSO is a specialization of an ESP, it can perform also energy-related operations, possibly on a smaller scale. For example, to satisfy legal constraints on the quotas of distributed energy that are derived from renewable sources, the DSO can select groups of FCs to buy renewable-energy credits and manage the "last-mile" infrastructure needed to collect the electricity into the existing power grid.

The FC is regarded in conjunction with the meter installed at his facility. He does not directly access any kind of data, as he can subscribe/unsubscribe supply contracts and require periodic reports about his consumption. Indirectly he sends measurements through the meters and receives billings. The FC consumption can also be profiled to offer him the most suitable contract. However, customer profiling should be done carefully, as it has been proven that profiling the consumptions of a household with a high time precision allows one to infer whether specific house appliances were active at a certain time. This information can also be exploited to infer the habits of the customer, thus representing a possible breach of his privacy. For these reasons, customer profiling should be limited to a time scale loose enough to prevent the leakage of private information.

The meter is the smart device used for the measurement of consumptions at local sites. It periodically sends consumption and technical data to the DSO. Consumption data are used for billing and technical data are used for load balancing. It can be connected or disconnected to/from the power line; it signals its operation and its faults to the DSO. It communicates over a secure channel with the DSO using symmetric cryptography, time-stamps and signature, in order to authenticate the DSO, and get authenticated.

A refinement of smart metering is that an FC may act as a local Provider of some resource type, usually electric power, e.g., through photovoltaic power generation. In this case he will have some of the provider/distribution services, to an extent limited by the small scale of his resource generation capability.

3.3 Security and Privacy Challenges

The design and deployment of smart meters raises several serious security and privacy issues, with different social and regulation concerns [1, 8].

A particular matter of interest for industry representatives and associations lies in the identification, assessment, and prioritization of the risks due to the potential frauds made possible by the deployment of devices with a security vulnerability [3, 17]. Indeed, if meter readings can be manipulated, either by returning false readings from credit meters or by forging authorization messages to prepayment meters, this could lead to substantial economic losses [15]. Thus, the minimization, monitoring, and control of the probability and/or impact of such unfortunate events is of great interest. In addition, at the level of the energy grid distribution, the presence of a remote off switch in all electricity meters can lead to a strategic vulnerability with respect to a capable cyber-adversary [18].

Considering the equipment suppliers, the main observation focuses on the excessive technical regulation sprung from the smart grid adoption, which threatens to drive up equipment costs in exchange for a small benefit [20, 23]. On the one side, the challenge of over-regulation leads to pessimism about the prospect of fixing security by mandating standards coming from a single authority. Moreover, the lack of universal standards for communications between meters and appliances might prevent the benefits of demand reduction being realized, as well as prevent reducing interoperability and competition [14, 25].

At the jurisdictional and organizational levels, it is possible to spot severe conflicts of interest. Indeed, the main goal of the governments is to cut energy use, which they hope to achieve by making energy use more salient to the consumers, while in most countries the meters will be controlled by energy retailers who want to maximize sales and who depend on pricing confusion. Meanwhile, the competition authorities should worry about whether giving energy retailers vast amounts of data about the customers, will adversely impact competition via increased lock-in [7, 16].

Finally, from the point of view of privacy activists, the main concern on the wide adoption of smart metering technology relates to the amount of sensitive personal information about the household usage that could be disclosed to principals able to access fine-grained consumption data [19, 22].

3.3.1 Security Engineering Requirements

From an engineering perspective, the security requirements for smart metering systems can be defined in terms of the three fundamental properties warranted to the data by secure systems: *confidentiality*, *integrity* and *authentication*. *Confidentiality* implies that a data stream, being sent from an actor to another one, should be readable only by the actors involved in the communication, and should not be eavesdropped by anyone else. This property can be warranted by means of tamper

proof communication endpoints, to avoid the insertion of eavesdropper devices, and through employing symmetric cryptography to avoid possible eavesdropping on the communication mean. *Integrity* concerns the need for the transmitted data to be delivered to the recipient in whole and unmodified. Integrity may be provided via a tamper evidence mechanism, such as a message digest, coupled with an on-failure re-transmission protocol. In order to properly warrant integrity, the message digest should change radically even when parts as small as a single bit of the transmitted message are changed, and it should be computationally unfeasible to forge a message with a valid digest, different from the original one. *Authentication* concerns the possibility of identifying either the author of a data block or, analogously, of finding the identity of the other endpoint of a communication. The most common means to enforce authenticated communications and to authenticate data is to employ asymmetric key cryptography coupled with a Public Key Infrastructure (PKI) able to certify the authenticity of the public key. Through these means it is possible to provide a secure, mutually authenticated communication between two entities, or to digitally sign data and applications, so that their authorship is undeniably traceable.

The aforementioned properties provide guidelines on how the various smart grid actors securely communicate over a channel, and how they store information in a database (where it exists – typically at a provider site). Long-term data storage ought to be restricted to the provider only. The energy provider is considered to be a trusted authority, able to keep its own perimeter free from attacks. Security issues may arise when considering inter-provider adversarial relations, where a provider may cheat spoofing the others' identity to obtain economic gains. However, since the providers' reputation is effectively a company asset, such threats are unlikely to get transformed into practical attack actions. As the providers should be able to intercommunicate between themselves, a proper structure must be deployed to render the asymmetric key infrastructure interoperable among all of them. To this end, either a common PKI should be established for all of them, or each energy provider should accept certificates signed by the others. Analogously, the DSO should be regarded as a trusted entity. Since it is difficult to access the power distribution structures, the security threats concerning the tampering with their metering and information concentration infrastructure, receive implicit mitigation by the safety measures included in the step-down power stations where they are placed. On the other hand, since the communication with the power meters happens through the common low-voltage power line, such a connection has to be secured properly. Basically the threats to a DSO are related to network attacks against the flow of data from meters and towards upper levels: user impersonation, rogue server hijacking the traffic, and denials of service caused by artificial message floods. This threat class is the one commonly associated with the security issues of communication and application network protocols. Most of the threats are on the consumer side. The consumer may in fact try to lead an attack against the meters. Typically such attacks are of the following types: physical tampering, side-channel analysis, network attacks against the flow of data towards upper levels, user impersonation, and connection hijacking. Also, eavesdropping attacks may cause

privacy loss in case a customer is able to gain information regarding the behavior of others. Authentication must be provided for the meters that have the ability to detach the consumer from the energy-providing grid, in order to prevent the unauthorized detachment of a single consumer or even a large scale intentional blackout targeted to cause massive disruption. Data collected from consumers should be aggregated for statistical purposes only, with no access to individual records in order to prevent fine grained privacy leakage from consumer profiling.

The aforementioned threats may be the result of a direct attack on the DSO infrastructure or, quite more likely, of a manipulation of the metering devices. In this respect, it is fundamental for the DSO to design and deploy secure meters that effectively hinder any possible malicious action by either outsiders or regular line subscribers. This goal can be achieved in two ways: either the DSO considers its own meter as a closed system where no changes to the running software (other than maintenance updates) are made, or the DSO employs the meter as an open system, allowing the customization of particular features by the line subscriber, via ad-hoc designed applications.

The first model assumes that the meter is realized as a closed embedded system, and deals with the confidentiality issues relying only on a shared secret with the DSO, which is embedded at manufacturing time. This in turn implies that the security margin provided by the infrastructure is based on the use of symmetric ciphers in order to wholly encrypt all the communications, thus providing complete confidentiality. The software maintenance updates are sent in encrypted form employing the same shared secret, without the burden of a complete PKI.

The alternative system implies that the owner of the meters is willing to run foreign, albeit certified, software on his own devices. In order to avoid the introduction of ad-hoc malware similar to recent SCADA-oriented viruses aimed at altering the measurements and/or the billing features, it is thus mandatory to employ a secure authentication infrastructure for the programs. This fully authenticated chain of trust must thus start providing authenticity warranties on the software components from the first phases of the boot, throughout the whole working cycle of the system. As this infrastructure is designed to foster collaboration and interoperability among the software produced by different meter manufacturers and smart grid stakeholders, it would be a welcome development to design common standards and criteria to provide a common platform on which to develop.

Similar efforts have already been born, and grown to a mature state for general-purpose computing: a known instance of such a consortium is the Trusted Computing Group (TCG) [26], which has built common grounds for personal computing endowed with a secure boot and chain-of-trust, realized through a specifically designed secure hardware module.

Analogously to the security issues tackled for the software, it is equally important to address hardware security issues. As a first step, the hardware components involved in a trusted system should be able to mutually authenticate, in order to avoid the insertion of rogue chips, or the bypass of critical validation components. As this is a common practice in unprotected systems, this threat should be properly addressed, as a hardware security breach results in immediate loss of trust for the

whole stack of software applications running on it. These concerns can be addressed via properly designed secure hardware modules, employing cryptographically strong primitives and tamper resistant enclosures.

In addition to the choice of the primitives and the enclosure design, another fundamental aspect to be addressed is to design side-channel attack resistant hardware, since this whole class of attacks is able to breach the security of a device without the need to interfere with its own tamper proof perimeter.

3.4 System Services

In the following tables, we will list the services provided by the various identified actors of the power grid. Security-related services are in italics. The "data" label refers to both Consumption and Technical data, unless differently specified. Services are reported in verb-noun form.

Table 3.1 reports the services related to the Meter actor, Table 3.2 those of the FC, Table 3.3 those of the DSO and Table 3.4 those of the ESP.

As reported in Table 3.1, the *Basic Meter* performs secure actions to manage data confidentiality during the transmission to the DSO. The complementary part to providing the required security margin of the meter actor is warranted by the tamper

Table 3.1 Meter actor and related services

Basic meter	Advanced meter
Compute and transmit billing data	Same services as basic meter and router
Compute and transmit consumption data	Programmability (e.g., upload certified applications)
Profile consumptions	*Trusted computing base – TCB or Trusted Execution Environment*
Report consumption and billing data to FC locally or to DSO/ESP remotely	
Manage data confidentiality, integrity and authentication (for data completeness, correctness and integrity)	
Ensure tamper-evidence and tamper resistance hardware	
Initialize crypto keys and/or certificates	

Table 3.2 Final customer services

FC services
Open/close the utility provisioning contract
Configure the home network (add/remove program, start-up/shut-down appliances)
Request a report to the meter (basic billing or network status or appliance status and consumption)

Table 3.3 DSO actor and related services

Single consumer or group (e.g., those living in a city area)	Group of appliances (group of appliances of the same type)
Measure	Report consumptions by consumer or typology (e.g., private user, company, department store, appliance)
Profile load	Control appliances (hours, times, tariffs, etc.) with policy to solve conflicts with FC
Report consumers' groups	*Initialize crypto-keys and/or certificates*
Aggregate data for profiling	
Control meter (e.g., diagnosis)	
Ensure tamper-evidence and tamper resistance hardware	
Manage confidentiality, authentication, and integrity of the data received from and sent to meters	
Manage the privacy of profile (aggregated) data	
Initialize crypto-keys and/or certificates	

Table 3.4 ESP actor and related services

Normal operation on the provider side	Normal operation on the consumer side
Manage power or resource in the grid	As the meter in a basic mode
Bill the FC	
Manage or use a PKI	

proof encasing, which can be endowed with breach detection sensors. Moreover, the secret keys employed for the communication should be properly stored in a volatile memory, which is erased upon intrusion into the meter box. In addition to the tamper proof casing and secure storage of the keys, the meter should also be designed in such a way that it is not possible to obtain the secret keys via measuring environmental parameters and exploiting the measures to conduct side channel analyses.

The metering device may be conveniently designed with extended functions to stream different types of multimedia contents inside a building, as well as to run custom programs provided by third parties [2]. To this end, as shown in Table 3.1, the resulting *Advanced Meter* should also include a full-trusted computing base compliant system [24]. Indeed, the possibility of adding custom programs besides the ones needed to securely perform the basic metering functions, implies that it is not possible to perform a building-time certification of the programs run on the device. Thus, the inclusion of a computing base able to validate the execution of trusted applications is justified by the offered programmability services.

The FC is able to access services via the facilities exposed from the smart meter. In particular, since the smart meter is running trusted software from the DSO and the ESP, the client can safely update the state of a provisioning contract without the need for extra paperwork. Moreover, in an advanced smart meter, the FC is also able to poll the meter in order to understand the distribution per-appliance of the power consumption of his house. Another service performable, thanks to the ability of the meter to communicate with other appliances, is to schedule the operation of power demanding appliances in time zones when the cost of the energy is lower, such as night-time.

The DSO will provide to the meter, and thus to the FC, the backend for all the services mentioned before. Consequentially, it should be able to perform periodic diagnostic operations on the transmission and distribution lines, employing the technical data collected by the meter in order to diagnose faulty or dissipating lines. Moreover, it should be able to propagate the consumption messages to the ESP in charge of billing a specific customer, thus providing data transport support for it. In addition to this, the DSO should be in charge of initializing all the key pairs employed to encrypt the technical messages to and from the final customers, the integrity of which must be warranted, lest they cheat on the payments. The DSO will also be employing the coalesced consumption data in order to avoid service interruptions due to peak requests by the FCs.

The ESP, similarly to the FC, may offer its services only through the support infrastructure provided by the DSOs, since DSOs ultimately act as data collectors and transporters. Normally, the behavior of the ESP only takes care of the billing and contract stipulation activities. During these activities, it may be required to set up an ESP-bound PKI in order to be able to digitally sign invoices for the final customer.

3.5 Standardization Activities and Related works

In this section we will recap the ongoing standardization efforts concerning the information security in smart grids. The National Institute of Standards and Technology (NIST) has developed security guidelines and a model of the power grid infrastructure for the US [25], which defines interfaces and implementation strategies for the smart power grid.

Analogously, the Zigbee Alliance [28] has now available a full-fledged wireless network solution, which has been successfully deployed in the US and can be used for both infra-meter and meter-to-DSO communications. As one of its defining features, ZigBee provides facilities for carrying out secure communications, protecting the establishment and the transport of cryptographic keys, and enciphering communications and controlling devices. It builds on the basic security framework defined in IEEE 802.15.4. This part of the architecture relies on the correct management of cryptographic keys and the correct implementation of methods and security policies.

The HomePlug Alliance [11], Wi-Fi Alliance [27], HomeGrid Forum [10] and ZigBee Alliance [28] have agreed to create a Consortium called Smart Energy Profile version 2.0 (SEP 2) [5] to enhance the interoperability among the standards and products of many organizations, whose technologies support communications over IP.

The PoweRline Intelligent Metering Evolution (PRIME) is an initiative driven by Iberdrola, the spanish DSO, for: "the definition and testing of a new public, open and non-proprietary communication architecture that supports remote meter processing functionalities" [21]. Its security proposal addresses security at low level through providing confidentiality, authentication and data integrity at the Medium Access Control (MAC) of the communicational architecture. Several security profiles are provided to deal with the different requirements of several types of networks. Confidentiality is guaranteed by encryption and from the fact that the encryption key is kept secret. Authentication is guaranteed by the fact that each node has its own secret key known only by the node itself. Data integrity is guaranteed by the fact that the CRC of the data payload of the communication is encrypted.

Individual EU Countries have begun to standardize common guidelines for tamper proof electronic devices, able to warrant the level of physical security required by meters. For example, the German authority (Bundesamt fur Sicherheit in der Informationstechnik – BSI) has released a dedicated guideline document for tamper proof smart meters [4]. The EU has instituted a task force aimed at analyzing and building recommendations for the future of the smart grid, and has released explicit guidelines regarding data protection and security in [8]. The open challenges related to the issues of privacy management and metering data aggregation are presented in [9]. In [19] the authors highlight privacy-related threats of smart metering and propose an infrastructure for secure measurements, which relies on trusted components outside of the meter. The authors in [22] propose a protocol that uses commitments and zero-knowledge proofs to privately derive and prove the correctness of bills, but that does not address aggregation across meters. Some techniques have been extended to protocols that provide differential privacy guarantees [6]. The technologies for smart grids, smart metering and more generally power line/wireless communications, are included in a large number of standards, each of which has a different scope of intervention and is only marginally related to security issues [7, 16].

Conclusion

As the need for energy increases constantly, the smart management of power grids has become a prime topic of interest for researchers and industry alike. In this chapter we provided an overview on the smart metering scenario and its novel information security challenges. We delineated the desired security services and the actors involved in the scenario, and provided a summary of the ongoing standardization efforts in this area.

References

1. Barenghi, A., Bertoni, G.M., Breveglieri, L., Fugini, M.G., Pelosi, G.: Smart metering in power grids: application scenarios and security. In: 1st IEEE PES Innovative Smart Grid Technologies (ISGT) Asia Conference IEEE, Perth (2011)
2. Barenghi, A., Breveglieri, L., Fugini, M.G., Pelosi, G.: Smart meters and home gateway scenarios. In: Cicchetti, A., Rossignoli, C. (eds.) IX Conference of the Italian Chapter of AIS Organization Change and Information Systems: Working and Living Together in New Ways, Roma. ITHUM Srl (2012)
3. Barenghi, A., Pelosi, G.: Security and privacy in smart grid infrastructures. In: Morvan, F., Tjoa, A.M., Wagner R. (eds.) DEXA Workshops, Toulouse, pp. 102–108. IEEE Computer Society (2011)
4. Bundesamt für Sicherheit in der Informationstechnik (BSI) – Federal Office for Information Security: Protection Profile for the Gateway of a Smart Metering System. https://www.bsi.bund.de/ (2014)
5. CSEP – Consortium for SEP 2 Interoperability: The Smart Energy Profile 2. http://www.csep.org/ (2014)
6. Danezis, G., Kohlweiss, M., Rial, A.: Differentially Private Billing with Rebates. Report MSR-TR-2011-10, Microsoft Research (2011)
7. Electronic Privacy Information Center: The smart grid and privacy. EPIC report, Washington, DC. http://epic.org/privacy/smartgrid/ (2014)
8. EU Commission Task Force for Smart Grids: Roles and Responsibilities of Actors involved in the Smart Grids Deployment. Expert Group 3 Deliverable. http://ec.europa.eu/energy/gas_electricity/ (2014)
9. Garcia, F.D., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: J.C. et al. (ed.) 6th Workshop on Security and Trust Management (STM 2010), London. Lecture Notes in Computer Science, vol. 6710, pp. 226–238. Springer (2011)
10. HomeGrid Forum: Gigabit Home Networking. http://www.homegridforum.org/ (2014)
11. HomePlug Powerline Alliance: Powerline Networking. https://www.homeplug.org/home/ (2014)
12. IEEE: Smart grid conceptual framework. IEEE Smart Grid Mag. (2014). http://smartgrid.ieee.org/ieee-smart-grid/smart-grid-conceptual-model
13. IEEE Computer Society: IEEE Standard for Information technology. IEEE Standard 802.11n-2009. http://standards.ieee.org/getieee802/download/802.11n-2009.pdf (2009)
14. Inter-operable Device Interface Specifications (IDIS) Industry Association: Inter-operability specifications. http://www.idis-association.com/ (2014)
15. Jawurek, M., Johns, M., Kerschbaum, F.: Plug-in privacy for Smart Metering billing. In: 11th Privacy Enhancing Technologies Symposium (PETS), Waterloo (2011)
16. Knyrim, R., Trieb, G.: Smart metering under EU data protection law. Intern. Data Priv. Law **1**(2), 121–128 (2011)
17. Liu, Y., Ning, P., Reiter, M.K.: False data injection attacks against state estimation in electric power grids. In: Proceedings of the 16th ACM conference on Computer and communications security, CCS'09, Chicago, pp. 21–32. ACM, New York (2009)
18. McDaniel, P., McLaughlin, S.: Security and privacy challenges in the smart grid. IEEE Secur. Priv. **7**, 75–77 (2009)
19. Molina-Markham, A., Shenoy, P., Fu, K., Cecchet, E., Irwin, D.: Private memoirs of a smart meter. In: Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building, Zurich, pp. 61–66. ACM (2010)
20. Petrlic, R.: A privacy-preserving Concept for Smart Grids. In: Sicherheit in Vernetzten Systemen: 18 DFN Workshop, Hamburg, pp. B1–B14. Books on Demand GmbH (2010)
21. PRIME Project: PHY, MAC and Convergence layers. PRIME Technology Whitepaper. http://www.iberdrola.es/webibd/gc/prod/en/doc/MAC_Spec_white_paper_1_0_080721.pdf (2008)

22. Rial, A., Danezis, G.: Privacy-preserving smart metering. Technical report MSR-TR-2010-150, Microsoft Research (2010)
23. Sander, K., Roos, B.: Security analysis of Dutch smart metering systems. Technical report of Universiteit Van Amsterdam. http://www.delaat.net/rp/2007-2008/p33/report.pdf (2008)
24. Teo, J.: Features and benefits of trusted computing. In: 2009 Information Security Curriculum Development Conference, InfoSecCD'09, Kennesaw, pp. 67–71. ACM, New York (2009)
25. The Smart Grid Interoperability Panel – NIST Cyber Security Working Group: NIST IR 7628 Rev. 1 – Guidelines for Smart Grid Cyber Security. Public report. http://csrc.nist.gov/publications/PubsNISTIRs.html (2013)
26. Trusted Computing Group: Trusted Computing: A Framework for Data and Network Security. http://www.trustedcomputinggroup.org/ (2014)
27. Wi-Fi Alliance: Wireless Local Area Network Interoperability Certification. http://www.wi-fi.org/ (2014)
28. ZigBee Alliance: Zigbee Smart Energy 2.0 Standard and Documentation. http://www.zigbee.org/Standards/Downloads.aspx#821 (2014)

Chapter 4
Authentication and Mutual Authentication

Asad Ali, François Tuot, and Gerald Maunier

Abstract In a sensitive environment, it is common to implement user authentication, possibly based on several factors, in order to ensure only authorized users have access to restricted features or information. But today more and more devices are interacting directly to perform some actions, or deliver a high level service to their user, like in smart grid and more generally in machine to machine environments. This raises the need for device authentication and, by extension, mutual device authentication. Highest confidence level should be achieved by ensuring both user and devices are allowed to engage in a transaction.

4.1 Basics of Authentication

Authentication is a mechanism of determining if an entity requiring access to a resource is the same he or it claims to be. This is a basic requirement for any system that manages access control. There are several methods to verify the authenticity of an entity involving credentials that can be divided into three broad categories: *what-you-know*; *what-you-have* and *what-you-are*. Even if some of these authentication means pertain more to persons (e.g. fingerprint) they all have more of less their equivalent for machines (e.g. PUFs).

A. Ali (✉)
Gemalto - Austin Ste 400, Arboretum Plaza II 9442 Capital of Texas, Hwy North, Austin, TX 78759 United States
e-mail: asad.ali@gemalto.com

F. Tuot
Gemalto - 6 rue de la Verrerie, CS 20001, Meudon Cedex, 92197, France
e-mail: Francois.Tuot@gemalto.com

G. Maunier
Gemalto - Avenue du Jujubier, Z.I Athelia IV, La Ciotat Cedex, 13705, France
e-mail: gerald.maunier@gemalto.com

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_4

4.1.1 What-You-Know

This category represents knowledge based authentication and is perhaps the oldest method of identifying users. Examples of such authentication methods are password & pass-phrase (entered directly on the keyboard or via various systems like joining discrete points of an image or encoding PIN), or answer to a security question that is selected by the user when an account is created, and then echoed back when access to that account is desired. The PIN for a smart card falls in the same category. The security of this method relies on the fact that only the owner has knowledge of the secret. In case this secret is compromised, so is the security of the authentication system (except for the smart card as the attacker also needs to have access to the card itself should the PIN be captured). Although this method is considered weak for transactions of high value it is still widely used by many businesses due to lack of a low cost alternatives. Authentications based solely on knowledge factor have some inherent shortcomings. Good passwords, such as long sequences of different character groups, are difficult to remember, while poor ones such as dictionary words or personal birthdays are easily compromised using various forms of software attacks. In addition, phishing and spoofing attacks may trick the user into providing the password directly to the attacker.

Systems can easily leverage passwords to authenticate to a remote party, but they of course need to be appropriately protected.

4.1.2 What-You-Have

Unlike the what-you-know methods that rely on a user remembering a secret, the what-you-have methods base their security on possession of a unique hardware token such as a smart card, USB token or a mobile device. This device is paired with the user, and can then perform some cryptographic computation on behalf of the user to attest his identity.

This kind of authentication method can be applied to machines, for example by using a TPM.

4.1.2.1 Public Key Cryptography

The public key cryptography is often based on what-you-have principal. In this asymmetric encryption scheme, a user's credential consists of two interrelated keys: a public key that is distributed to others with whom the user or system communicates, and a private key that is never disclosed to anyone. It is this private key which represents the what-you-have concept. To protect this private key, it is

often stored inside a hardware token which the user carries with him. Instead of distributing the public key directly, it is packaged inside a digital certificate, which is also called public key certificate. These certificates are issued by trusted third-parties called certificate authorities (CA) who confirm that a given public key belongs to a particular user or machine. The entire eco-system around generation, issuance, use, and revocation of keys is referred to as public key infrastructure (PKI).

The PKI certificate based authentication is one of the strongest authentications methods, especially when coupled with a secure element and stringent user verifications are enforced by the certification authority in charge of certificate issuance. It uses the classical PKI challenge-response handshake to verify a user's identity through a signature that is generated by the private key unique to this user. The strength of the PKI solution depends heavily on the extent to which the user's private key can be protected. The use of smart card to store this user key or a TPM to store a machine key is therefore a natural option and required to implement binding (or qualified) signature.

Examples of public key cryptography include SSL/TLS protocols that protect HTTPS web traffic, VPN that allows communication over private networks, and encryption as well as signature when sending secure eMail.

TPM attestations, as defined by TCG, are also based on PK cryptography to provide at the same time strong and reliable machine identity and some software configuration information to assess the system trustworthiness.

4.1.2.2 OTP

As the name suggests, an OTP is a one-time-password that can only be used once. If it is presented as proof of user's identity a second time, it is rejected. Generally OTP is a 6–8 digit number generated by a hardware device. This hardware device is either a dedicated OTP token the size of a USB mass storage stick, or a credit card size card. More recently, OTP can also be generated by an application on a smart phone, thereby eliminating the need to carry a dedicated token. Regardless of the device used for generating it, the OTP is generally combined with traditional username/password method to increase the strength of authentication. There are two broad categories of OTP generation algorithms: time based algorithms such as the one used with RSA SecureID tokens; and event-based algorithms such as that proposed by the Open Authentication (OATH) consortium. While the latter algorithm is an open standard, the former uses a proprietary technology. In both cases the OTP token is bound to a unique authentication server and have to be synchronized, so that OTP values generated by the token match what the server is expecting. This synchronization, or device provisioning as it is sometimes called, is a critical piece of all OTP based systems.

4.1.3 What-You-Are

This category places its security on the natural habits or biological characteristics of the user and even sometimes of the machine. The entity does not have to remember or store any secret nor carry any dedicated device. Authentication is done using what the user or system is or does.

4.1.3.1 Biometry

Biometry has a high-profile security image in the public. It is generally associated with sophisticated technology that uses verification of personal traits to authenticate a user. Such authentication methods not only increase convenience (compared to passwords) but also solve some inherent problems with tokens and passwords: they can be borrowed, lost or forgotten. The personal traits used in biometry range from fingerprints, face, voice, hand geometry or vein patterns. In some more specific contexts, secure systems may also use iris or retina scan. However, depending on cultural factors, such methods may be considered intrusive and therefore rejected by users. In addition, privacy laws in some jurisdictions may severely restrict or even forbid creation of central databases to collect such personal biometric attributes.

Biometric authentication, which is also called 1-to-1 matching or identity verification, consists of three steps:

1. Acquire the raw biometric data from a user. For example, producing an image from a fingers scan.
2. Extract characteristic points (also called *minutiae* in a *candidate template*) from the raw image.
3. Match these characteristic points with a model associated with user's identity (called a *reference template*).

The acquisition and extraction processes are also used during initial enrollment to build the reference template and associate it with the user's profile.

Although it may appear that the most sensitive operation for an authentication process is the matching phase, since it is responsible for providing a YES/NO result, in reality other elements in the chain like acquisition and reference template storage must also be protected. For example, an attacker trying to impersonate a legitimate user could submit a previously acquired finger image to the system or replace the reference template with his own template. Security implications greatly vary depending on the verification environment. While a user interacting with a system managed by security personnel may have few opportunities to tamper with the system, an un-manned automated authentication systems hosted locally or on the network may be subject to additional forms of attacks.

To better protect user's reference template the match can also be done on a secure element such as a smart card owned by the user. This is referred to as match-on-card (MoC), and since user is in control of his reference template it increases their confidence in the system. MoC can be used as PIN replacement for convenience or as another authentication factor in addition to the smart card PIN.

Biometrics also allows for identification of a user by matching an anonymous candidate template to an entire database of reference templates (1-to-n matching). This model is longer but useful in criminal investigations where the supposed owner of the candidate template is not known.

Due to the variations in acquisition environment (such as image angle, pressure on the sensor, etc.) and changes in the user characteristics (such as hurt or aging skin etc.), each data acquisition for a single user will be different. As such, biometric verification is always trying to determine "how closely" a raw image matches a reference template. Matching performance is a trade-off between two parameters: FAR (False Acceptance Rate) and FRR (False Rejection Rate). High security systems will raise the FAR so that it is less likely that an unauthorized user will gain access the system, while consumer market products like mobile phones or personal computers sensors will extend the FRR to ensure that legitimate user access will never be denied.

Since biometrics data is unstable by nature, it cannot be used for direct cryptographic operations. Cryptography requires stable data. For example a key "close" to the real key but off by only 1 bit is useless. Authentication systems therefore rely on biometrics to *grant access* to a key (bio-PIN). This key is then used for cryptographic operations.

What you are and what you have methods cannot be more different and are therefore complementary: whereas a knowledge based authentication such as PIN relies on a secret easy to personalize, change and delegate to someone, biometric traits are usually public, cannot be modified or delegated, and require heavy enrollment workflow. In addition matching algorithm is not obvious and acquisition may leave some traces (for example, finger image on a sensor).

4.1.3.2 Behavior

Behavior based authentication techniques allow for implicit authentication of users based on the pattern of their interaction with the device or computer. Examples of such patterns can be the speed with which a user types on keyboard when entering their username or password, or the pause between specific characters. For mobile touch screen devices that ask users to trace their login pattern the application may consider the stress points during the finger swipe. By their very nature behavior based techniques are not absolute in their determination of a valid credential. As such, they are always used in combination with other more discrete forms of authentication as knowledge or OTP. One advantage of behavior based approach is that is the least intrusive of all authentication methods. The user does not have

to remember or carry any dedicated token. He simply interacts with the device and the authentication system does the rest. Principals of behavior approach can also be applied to location based authentication, where the user is allowed access if he is connecting from his usual devices; for example from his office laptop connected through his office network. This behavior pattern is considered safe, while the same laptop when connected from a public network may not be safe, and thereby warrant additional authentication credentials.

4.1.3.3 Physical Unclonable Functions (PUFs)

Biometric and behavior based authentications verify unique characteristics, but this need not be limited only to humans. Hardware elements, even if produced with the same process on the same production line, have tiny differences: for example, non initialized memory state will be unique, as well as performance or glitches generated when going through gates. Like other biometrics processing, these differences are difficult to find and analyze as they are not stable over time or under environmental constraints.

Initially intended to detect counterfeiting of goods, PUFs can also be complemented by smart software algorithms to select stable and differencing bits to produce a unique – never stored – key from these unique traits. During the device manufacturing, this key may be extracted from the device itself and provisioned in an authentication system for subsequent remote device authentication. This key can then be used in several authentication protocols described in Chap. 6.

4.1.3.4 Device Fingerprinting

Even if not directly linked to user authentication, device fingerprinting is becoming an important part of the authentication process. Knowing that user credentials are not only valid but also coming from a previously encountered non-rooted device brings more confidence in the authentication result.

There is a multitude of device attributes (or device fingerprints) that can contribute to a device identity. These attributes include unique device identifiers, MAC addresses, memory size, OS version, the number of applications installed, the size of the contact list, and even the movements of the device.

4.1.4 Credential Delivery

The authentication credentials can be delivered from the user to the authentication system in a variety of ways. There are three basis mechanisms for this: local; in-band; out-of-band.

4.1.4.1 Local

The local credential delivery is used when both the client application accepting the credentials and the back-end system performing the match are on the same host. Examples of this approach are when we enter the PIN in a smart card reader, swipe our finger on mobile, or when logging into a PC.

4.1.4.2 In-Band

The in-band credential delivery is used when the user enters the credential into an application which will be the primary application for interfacing with the system after authentication. For example, if a user connects to a web server using a web browser, and enters the login credential in the same browser window, the credentials are entered in an in-band manner. Same classification applies to mutually authenticated TLS session established between a client and a server based on certificates.

4.1.4.3 Out-of-Band

The out-of-band authentication is done by using a separate communication channel from the one used for normal delivery of service content. For example, if a user connects to an Internet portal through a web browser running on a PC, the out-of-band channel can be through the user's cell phone. This duality of communication channels offers by design stronger authentication solution. The typical authentication flow can be as follows:

1. User enters login credentials (e.g. username and password) in the browser.
2. The login server validates these credentials and takes user to a second login step, where user is asked to enter a numeric code.
3. The user reads the code from his phone, and enters it in the browser.
4. The server validates this code and authentication completes.

The numeric code entered by the user in step 3 is generally a six to eight digit number. It is either sent to the mobile device from the authentication server or generated on the mobile device itself in response to a trigger from the server. In the latter case, the code generation can be done by a secure element embedded in the device or by the software running on the device. If code is generated on the device, there has to be a pre-registration of the device with the authentication server to synchronize the algorithm. An example of server generated code is the extended authentication offered by internet service providers like Dropbox, Google and Facebook. With the widespread use of SMS capable phones, this is becoming a low cost approach for enhancing security of online authentication. Several online service providers are offering this option to their users. An OTP application running on the phone is an example of device generated code.

An even stronger form of this out-of-band approach is to use the SIM card or software inside the handset and perform a cryptographic exchange directly with the authentication server. For example, Valimo offers a PKI based mechanism where the user's private key in the SIM signs a challenge from the authentication server, and thus validates the user without any manual interaction with the primary access device.

4.1.5 Method Strength

Before we can decide which authentication method to use when, we have to be aware of the relative strengths of these authentication methods. While there is no standard benchmark for this, the general consensus in the industry is that individual authentication methods can be ranked in their level of security through a combination of the method itself and context in which the credentials are used. This context is determined by various factors such as location of credential storage or generation of one-time credentials. For example PKI credentials stored in a secure element are more secure than the ones stored in software. Furthermore, secure elements that are tightly embedded to a host device are generally considered less secure than the secure elements which can be detached. A smart card that is connected to a PC when required but is otherwise in the physical possession of the holder offers greater security as well as assurance.

4.2 Mutual Authentication

In most of authentication models, only the client (or user) willing to consume services from a server (resp. a local machine) has to prove its identity with one of the techniques presented previously in this chapter to be granted access.

But sometimes security considerations imply both entities want to establish the trust level of the other peer before exchanging data or providing a service. Typical examples are an employee using a web client to connect to a corporate web server, or two devices willing to exchange data.

The latter model was illustrated in eniac funded TOISE project, with a secure token granting access to user private storage only after proper authentication of the host machine – which has to be previously registered –, and respectively the host machine forbidding removable storage devices unless they have been previously enrolled (Fig. 4.1).

Such mechanism put additional requirements on the authentication methods, so that the authentication process doesn't leak any sensitive data. For example, a fraudulent system interacting with a legitimate entity must not receive any credentials allowing impersonation of the legitimate entity in an interaction with the other legitimate party. For example a password transmitted in the clear or a

Fig. 4.1 USB token is typically provided by the IT department of a company to an employee, and used for user authentication and access control to various resources of the infrastructure (application servers, databases, etc.)

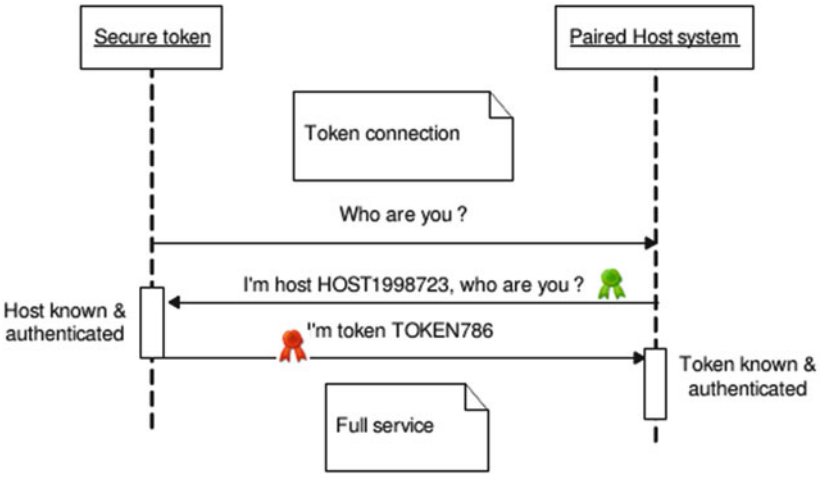


Fig. 4.2 After token connection, each party must receive proof that other party is legitimate to provide nominal service

fingerprint candidate template cannot be used to bootstrap such mechanism. Most of the time, challenge/response mechanism is the answer to derive key materials specific for a session and not disclosing any sensitive information. The two-way TLS and IEEE1667 are examples of specifications allowing mutual authentication models (Fig. 4.2).

Chapter 5
Low Power Wireless Sensor Networks: Secure Applications and Remote Distribution of FW Updates with Key Management on WSN

Juan Rico, Juan Sancho, Álvaro Díaz, Javier González, Pablo Sánchez, Bibiana Lorente Alvarez, Luis Andres Cardona Cardona, and Carles Ferrer Ramis

Abstract Nearly 70 % of the average household utility bill could be influenced by Wireless Sensor Network (WSN) application to temperature and lighting. Home and building owners can use interactive energy management tools to create energy management profiles that are triggered by certain established consumption rates. However, security remains a critical issue. In this chapter, we will present how the new security features (both on hardware and software level) introduces improvements in the overall WSN picture, and we will explain with detail the new OTAP procedure. Furthermore, we will state all the new possibilities opened for Low Power WSN based on the enhancement of security, being a good candidate for applications restricted to much more powerful devices.

J. Rico (✉) • J. Sancho
TST, Calle Albert Einstein 12, 39011 Santander, Spain
e-mail: jrico@tst-sistemas.es; jsancho@tst-sistemas.es

Á. Díaz • J. González • P. Sánchez
University of Cantabria, ETSIIT, Avda. de los Castros, 39005 Santander, Spain
e-mail: adiaz@teisa.unican.es; javiergb@teisa.unican.es; sanchez@teisa.unican.es

B.L. Alvarez • L.A.C. Cardona
CNM-IMB (CSIC), Centro Nacional de Microelectrónica – Instituto de Microelectrónica de Barcelona, Consejo Superior de Investigaciones Científicas, Campus de la Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain
e-mail: bibiana.lorente@imb-cnm.csic.es; andres.cardona@imb-cnm.csic.es

C.F. Ramis
CNM-IMB (CSIC), Centro Nacional de Microelectrónica – Instituto de Microelectrónica de Barcelona, Consejo Superior de Investigaciones Científicas and Department de Microelectrónica i Sistemes Electrònics, Universitat Autònoma de Barcelona (UAB), Bellaterra (Barcelona), Spain
e-mail: carles.ferrer@imb-cnm.csic.es; carles.ferrer@uab.cat

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_5

5.1 Introduction

Low Power Wireless Sensor Networks (WSNs) consists of spatially distributed self-powered sensing device to monitor physical or environmental conditions, such as temperature, pressure, sound, vibration, motion etc. and to cooperatively send their data using a wireless medium. WSNs have application to a wide range of areas including military, health care, security and industrial, this project focuses on environmental and infrastructure monitoring. Commonly, this kind of network (depicted in Fig. 5.1) consists of a large number of low-cost, low-power, resource-constrained sensor nodes operating often in an unattended, hostile environment, with limited computational and sensing capabilities.

These deployments offer different functions and capabilities that must be secured in order to reduce the vulnerability of the network itself and also the risks at device level. In order to provide a robustness solution there are several requirements at different levels that characterize these networks. Traditional security schemes cannot be applied to these networks due to the particularities of devices participating in Low Power WSN.

The security features are usually associated to higher power consumption. On the one hand is directly related to the overhead introduced in data packets and control mechanisms, on the other hand, this also requires higher computational capabilities. In this context, node reprogramming is a critical task since the data sent through the wireless network defines the behavior of devices, thus it is mandatory to protect information and procedures and provide enough robustness to the system for not allowing malicious firmware to be installed in the system.

The basis of Over The Air Programming (OTAP) is the distribution of packets containing parts of the firmware to be installed in certain nodes, then the assembly them at node level recomposing the whole firmware image, and finally, rebooting the devices for running the new software version.

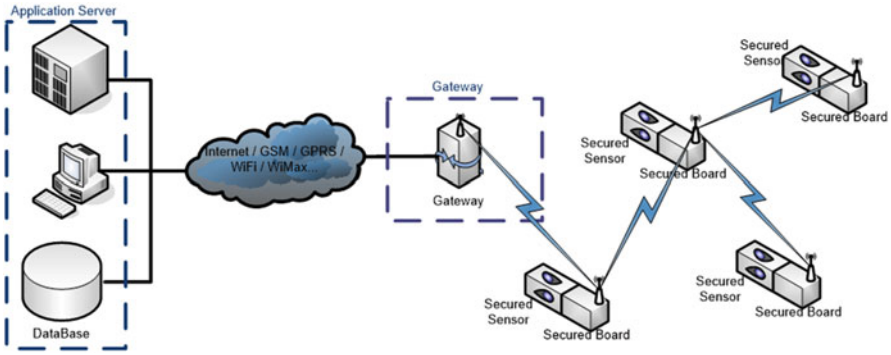


Fig. 5.1 Low power WSN architecture

5.2 Secure OTAP

5.2.1 Introduction

Secure OTAP protocol aims at reducing power consumption and increase security level in this crucial task for meshed wireless networks. The design of the protocol has been done considering the benefits of existing solutions such as [19, 21, 22, 34], but focusing also in overcoming the difficulties associated to battery powered devices. The protocol proposed enables the update of devices in an efficient way without exposing devices or information to threats and risks.

Figure 5.2 presents the flow in the execution of the protocol. Firstly, it is necessary to provide a new version of the software, once this is ready, it has to be uploaded to a server which centralized the distribution to the different networks where this version has to be installed. At this point, next step is the dissemination of the new firmware among the different elements that are target and want to install it. And finally reboot devices running the new version. The following sections describe the internal mechanisms of each of these phases.

5.2.2 Actors

The protocol identifies three main actors setting the pace in the update procedure. Each of them is different and represent data source, data destination and communication link paradigm. The data source is the piece of the architecture where the new

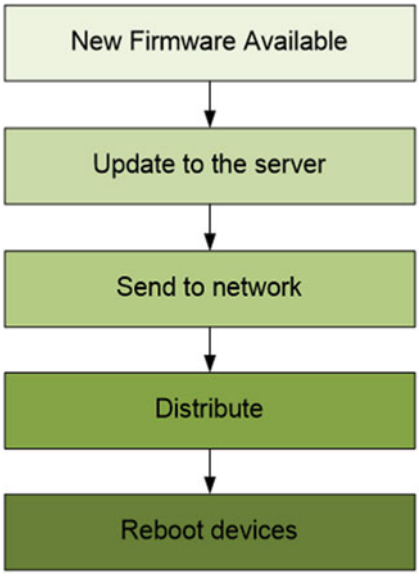


Fig. 5.2 Secure OTAP phases

firmware is stored, in this case the OTAP server. This communicates to the network through OTAP gateways, the bridge information from the servers to end devices. A deeper explanation of each element follows:

- OTAP server: It is owned by the devices manufacturer or derived to the WSN operator. This server manages the firmware version that should be running in each device and the one which is currently running. Since it is the source of new version is the element in charge of controlling the flow of the whole OTAP process.
- Gateway: The gateway acts as bridge between the public IP network and the Low power meshed networks where the devices that are going to be updated operate. As consequence, it enables the communication between the OTAP server and end devices. Since gateways are not constrained devices, some of the functions managed by the server can be delegated to the gateway so as to reduce traffic in the network and also for simplicity of the processes. By the way, the delegation of functions into the gateway does not expose the network to higher threats and vulnerabilities.
- End Device: The final destination of the firmware distribution. Each End Device decides by a set of dynamic rules if participating in the firmware distribution so to receive the new program image, installing it and rebooting accordingly. It also implements a kind of backup program image so to recover itself in case a problem occurs.

As presented in Fig. 5.3, the three elements cover different part of the OTAP procedure, thus the requirements and activities performed in each of the elements are different.

5.2.3 Protocol Messages

Once presented the elements that take part in the remote update process, the next step is to introduce the elements of the protocol. The OTAP protocol is composed by two main groups of messages, the ones containing pieces of the new software version, and a second group of control messages. The first are the software itself, and the second are the control messages required for introducing security and preventing attacks.

5.2.3.1 Firmware

The main element in the OTAP procedure is the firmware that should be installed in the destination End Devices. Due to network constrains, the firmware is divided into pages each of them univocally identified so as to allow destination to recompose the original firmware even receiving the chunks in different order.

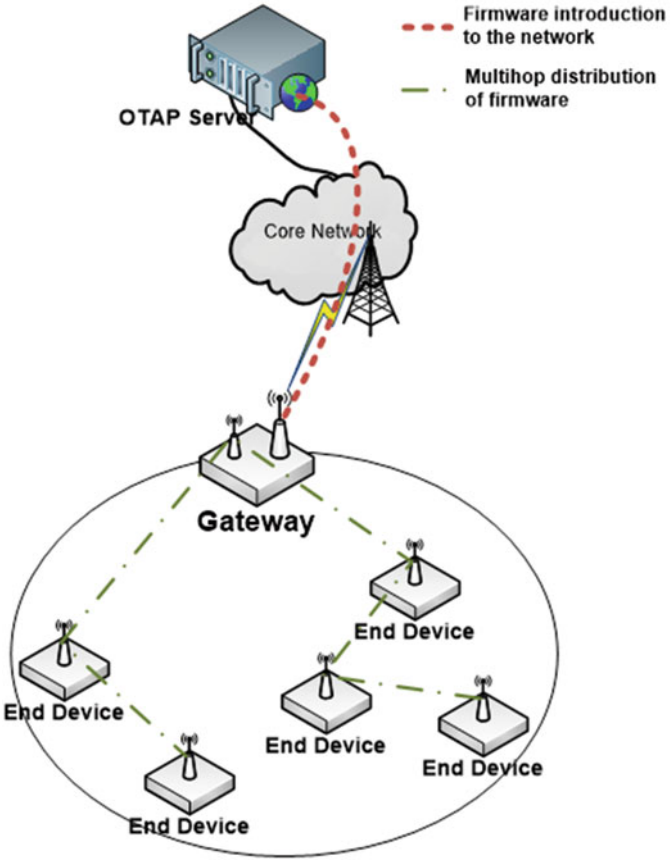


Fig. 5.3 OTAP architecture

The split in pages also allows adding advanced cryptography information to some of them, thus demanding a high knowledge of the network to attackers in order to get information from sniffing the channel. The key management for encryption and decryption is out of the scope of the protocol, nevertheless the protocol is designed so as to allow the inclusion of such advanced techniques.

Figure 5.4 shows the partition of a complete firmware version into pages. The size of the page is adaptable to a size suitable for the underlying radio technology. These pages represent the core of the data packets, but it is also mandatory to include some control information so as to allow the disordered reception of data packets. The final frame is based on TLV [29] format and it is presented in the following figure.

As depicted in Fig. 5.5 the key of the frame is the Page number and Checksum fields, the first allows to allocate in the right place in the memory the binary part received, the latest is used for assuring the validity of data received after decipher process.

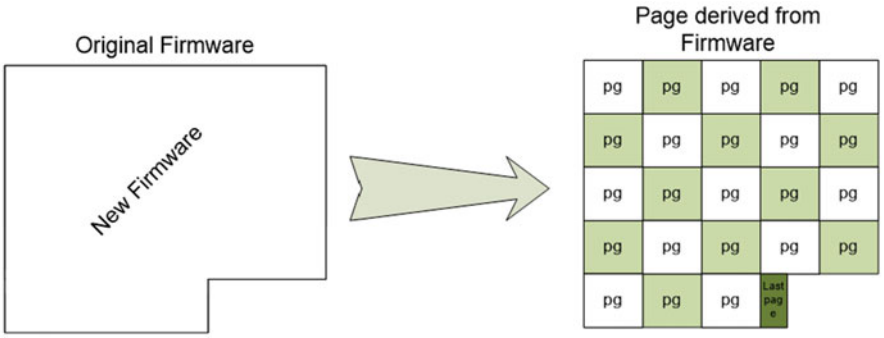


Fig. 5.4 Firmware split into pages

Fig. 5.5 Data frame format

It is also important to note that ciphering the data introduces a new security level. The key management scheme is open and complementary to the OTAP protocol. The restrictions are set by the technology MTU and the computational capacity of the devices participating in the network. Furthermore, so as to reduce consumption it is possible to cipher some of the pages.

5.2.3.2 Control Information

The key part is the distribution of the firmware, but in order to make it as efficient as possible, there are several procedures that are required for defining who are participating in the update process, and whether they are actuating properly or not. Some messages enable the flow control of the OTAP, but the key one is the first sent. This packet contains the so called metadata. Most definitions of the term agree that it refers to data about data, data about data content and content about content.

Figure 5.6 introduces the necessary information included in the metadata message to identify a unique *Program Image*:

- Firmware ID: the first field of the metadata structure is used to uniquely identify a Program Image.
- Target ID: the second field points out the group of End Devices which will be participating in this firmware update. Considering that a network is composed by heterogeneous devices performing different actions, it is necessary to check the suitability of the nodes for updating to an specific version.

Fig. 5.6 Metadata structure

- Page Info Structure: this field collects all relevant info related to page distribution, mainly page size, number of pages and type of encryption in use. Due to the adaptability of the protocol this also introduces a limit randomness factor that can be also exploited for increasing network robustness. This field also enables the preparation of the memory portion where the new firmware is going to be stored. The knowledge of page size and number of pages is enough for reserving the memory that is going to be demanded by the new firmware version.

5.2.4 OTAP Information Dissemination

The key part where most of the energy is consumed in a OTAP/MOTAP is due to the activity of radio interfaces. Since traditional protocol have not considered the requirements and limitations imposed by constrained devices, there are several improvements to be applied at this point of the procedure. This section gives a step by step insight of the whole protocol, defining the activities carried out by each actor in each phase and explaining the messages exchanged by them. A quick view of the protocol phases is shown in Fig. 5.7.

- **Phase 1:** Defining the list of nodes that will take part in the update process.
- **Phase 2:** Dissemination of information so as to provide the correct binary file to the nodes in the process.
- **Phase 3:** Rebooting devices with the new firmware version running.

5.2.4.1 Serving Firmware Version from OTAP Server to Gateway

The protocol assumes that the OTAP server is offering a secure method for uploading new version of software. This new version requires several parameters such as *firmwareID* and *targetID* so as to automatically generate the metadata and reduce this way the operations to be done in the weak part of the whole chain.

The exchanged messages corresponding to this step are shown in Fig. 5.8. It details the Program Image transfer sequence from the OTAP Server to the Gateway. The server sends the metadata to the gateway, this check if it has been already

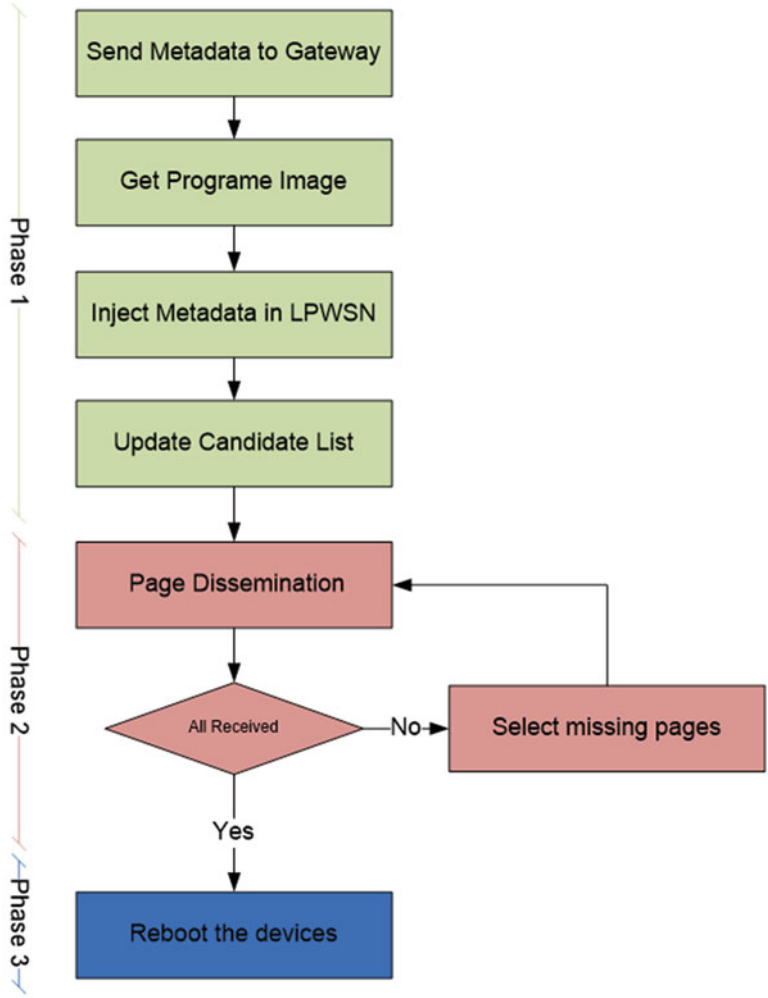


Fig. 5.7 OTAP protocol phases and flow

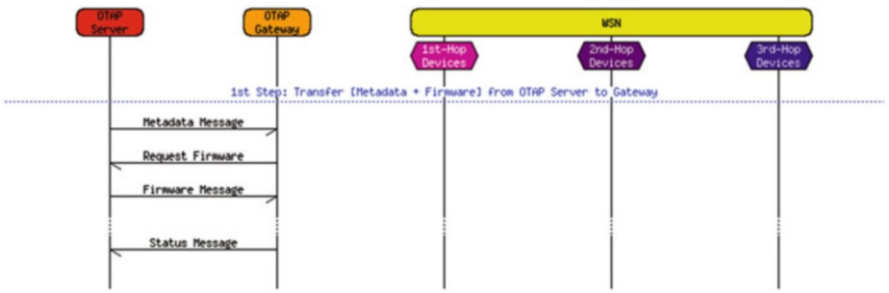


Fig. 5.8 Distribution of software from OTAP Server to Gateway

spread in the network behind it, in that case it does not do anything. In case it is a new version not already distributed to the network, it requests the firmware so as to locally store this new software.

Considering the typical case, due to server intelligence prevent non-sense use of resource, the typical flow for this part is the following. The first message contains solely the Metadata, already presented in previous section, and it is sent from OTAP Server to Gateway. Its main purpose is to inform the Gateway about the availability of a new Program Image ready for dissemination. The Gateway answers back the OTAP Server with a Request Firmware message, in case it is necessary. Finally, the OTAP Server starts to send the binary file (the pages). Once done, the Gateway proceeds to check the integrity of the received information, for instance computing the CRC or checking file authenticity by deciphering the pages. This step finishes sending back an ACK Message to the OTAP Server.

5.2.4.2 Disseminating Program Image into WSN

The previous phase is quite simple due to the number of constraints and limits are really low compared with the dissemination in the Low Power Wireless Sensor Network. The protocol allows multi-hop distribution so it makes really important the efficient use of spectrum by sending messages in the most effective way.

Although the information is stored in the gateway the whole process is ruled by the OTAP Server, none of the activities start without its command. At the end and forced by these networks particularities, it is extremely important to trigger the dissemination in a period that does not imply potential problems to the devices.

The dissemination process is started by the OTAP Server sending to the *Gateway* an *Injection Message* indicating that a *Metadata Message* must be delivered to all the devices in the whole network. Different works [39] have demonstrated that pure flooding is the best choice especially in sparse networks, so the *Metadata Message* is broadcasted to the whole WSN (Fig. 5.9).

Upon reception of the *Metadata Message*, a specific task running in the *End-Devices* filters out OTAP messages from any other kind of application messages. The End Devices process the information in the Metadata Message thus being aware whether the new software is suitable for them or not. More precisely, an End-Device becomes *Candidate* verifying that:

- Firmware ID received is newer than running one
- Matching Target ID with End-Device's internal ID
- The security level of the Metadata Message is compatible with the End Device

If these three rules are positive the End Device sends a *Candidate Message* to the **Gateway** including the corresponding Firmware ID. If any of them is negative, End-Devices may send a *Non-candidate Message* in order to inform the **Gateway** that despite having received the *Metadata Message*, they are not involved in the OTAP update process for this specific version. Making use of the third parameter (Page Info Structure), *Candidates* make room in memory for page storage. Note that

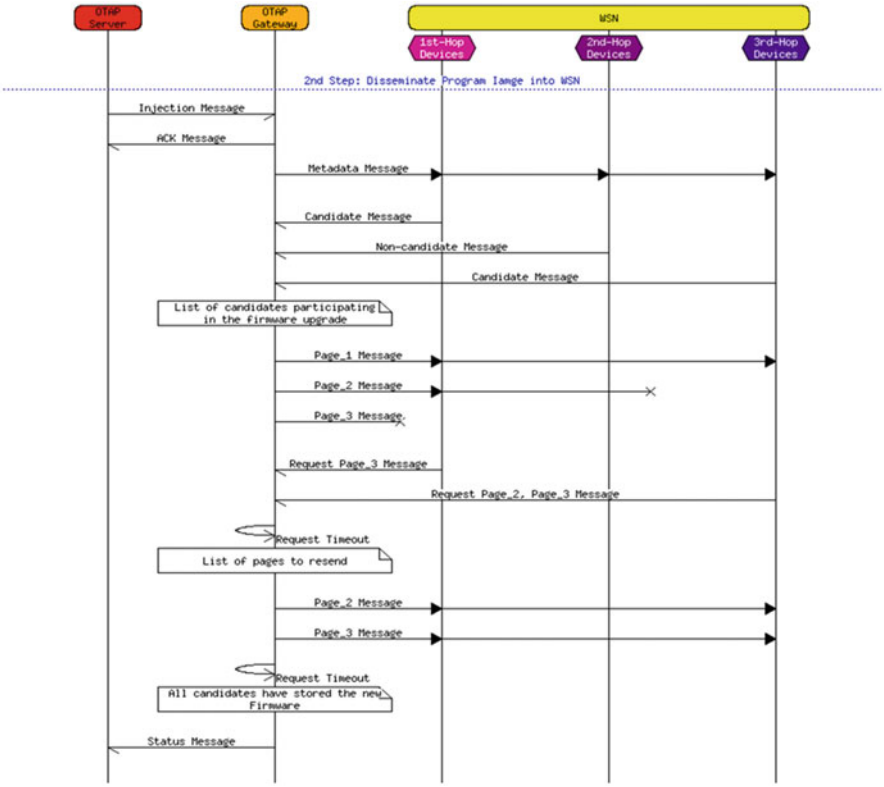


Fig. 5.9 Distribution procedure of software in a multi-hop WSN

the OTAP mechanism offers the ability to configure if *Non-candidates* shouldn't send a *Non-candidate Message*. This configuration is a must when communication efficiency is important in large scale WSNs, because it avoids flooding the network with messages that can be omitted and substituted by a timeout. This configuration improves network performance by reducing traffic and increase energy efficiency by avoiding the re-transmission of many packets.

After an application timeout, the **Gateway** would have compiled a list of *Candidates* participating in the OTAP update. Even though there are some end-devices in the LPWSN that are not taking part of the OTAP update, the **Gateway** broadcasts every page. Later, candidates store the received pages while compiling a list of missing pages. For each page received integrity and authenticity verification is done (depending of the security level selected by the system administrator). Once all pages are successfully received an *Acknowledgement Message* is sent out to the **Gateway**. After an application timeout, if candidates didn't received all the expected pages they send a *Request Pages Message* in order to ask for the **Gateway** to resend those missing pages. Meanwhile, the **Gateway** is compiling a list of missing pages for each *Request Pages Message* received and after a while it re-sends again all those

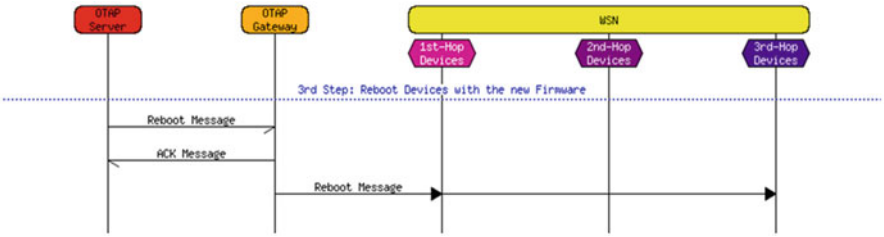


Fig. 5.10 Rebooting command for finishing OTAP

missing pages. Broadcast or Unicast transmission is done depending on the number of *Candidates* requesting for the same missing pages. A configurable threshold may control that.

The Program Image dissemination process iterates until no more requests are received. To finish this step the **Gateway** sends an *Acknowledgement Message* to the **OTAP Server**, which includes a report about the status of the operation and the list of *Candidates* ready to update their running application.

5.2.4.3 Rebooting with New Firmware Version

The last step in the reprogramming procedure consists in stop the outdated version and start running the new one. In order to do so, since the binary file is already stored in device memory, the **OTAP Server** sends a *Reboot message* which includes a code created using the Firmware ID, the Target ID and the list of candidates. This way it is possible to provide an ID that matches only those candidates ready to proceed with the update. Additional rebooting codes could be created for special firmware ID's, as for example for rebooting the *Recovery Image* of a set of End-Devices (Fig. 5.10).

Upon reception, the End-Device stops running the original application, it saves some variables in backup registers if needed and proceeds to execute the Secure Booting application. This last step of the OTAP mechanism finishes without acknowledgment messages by default, although is possible to force a message to be sent from specific key end-devices to the OTAP Server.

5.3 OTAP Partial Firmware Update

5.3.1 Introduction

Wireless communications are exposed to a great variety of threats and attacks. Node reprogramming is a critical task since the data sent through the wireless network defines the behavior of the devices, thus it is mandatory to protect this procedure. The goal is to provide enough robustness to the firmware update to avoid installing malicious firmware.

The OTAP is composed of three stages or steps: distribution of the packets containing parts of the firmware to be installed, the assembly of the packets at node level recomposing the whole firmware image and, finally, the rebooting of the devices so the new firmware is executed. The proposal presented in this section defines a simple but effective methodology to reduce the power consumption of the firmware over the air update process while maintaining a secure transmission.

It is important to explain that the non-volatile memory available in the node is usually a FLASH-type memory of small size (around 16 MB). This memory is divided in blocks or pages and it is necessary to erase each block before writing for the first time. In addition, it is necessary to erase the complete block before writing a previously written memory position of the FLASH memory block. This erase and writing process increases the execution time of the update process, the power consumption of the node and reduces the life of the memory. Thus, it is very important to manage it adequately.

There are just a few techniques in the literature related to firmware update in WSN. These methodologies are categorized into different classes, from full-image replacement to incremental updates. Related to the full-image replacement case is possible to find several articles. In [13, 43] and [35], the authors present different protocols to update the complete firmware image. These methodologies replace the entire firmware image including applications, operating systems and drivers. However, they are not efficient because a little change in the firmware implies the whole firmware replacement, and this requires a high amount of energy consumption. To solve this problem, some authors proposed in [14] and [30] to use incremental reprogramming methodologies. These protocols divide the firmware into blocks. And only the blocks that are modified are transmitted. So, they reduce the power consumption of the firmware update process comparing whit the full-image protocols because they do not transmit the complete firmware image. However, they have a problem that in this document is called “code shifting”. In the case of just a little change in the firmware (for example, with the inclusion of a new function or a function size modification), the whole set of functions and instructions can change their memory position. Therefore, all function addresses must be replaced to match the new memory location in all of the jump instructions and function calls. This implies that even a small change in the firmware can cause a high amount of code modifications.

In [17], the authors try to solve this problem by inserting a “slop space” between firmware functions. This “slop space” allows the growth of any function without affecting the position of the functions available after the modified function. Therefore, it is possible to rewrite the new function in the same memory position. However, in the case that the new function is bigger than the “slop space”, the problem still appears.

There are other update techniques such as the use of a virtual machine. In [3, 7, 18, 20] is proposed to use this virtualization technology. The problems of these schemes are the complexity of the programs and that the power consumption during the just-in-time compilation linking or execution process is higher than the use of a native code. For example, Mate is presented in [20] as a compact virtual

machine designed specifically for WSNs. It contains support for code updating but its associated energy overhead is prohibitive. In addition, this tool only supports applications updates and not other specific parts of the firmware.

Other techniques are based on the use of dynamic linking. According to [12] is possible to load dynamically individual software modules in the node. This technique uses position independent code. The software modules communicate with the kernel via a system jump table.

Other approach was introduced in [25]. In this paper the authors present a combination of two techniques: the use of indirect functions calls (code independent of the position) and the use of the differences between the firmware versions to create an update script. The improvement achieved with this technique is that the size of the update script is reduced. However, it still has the problem of the dependency between functions. If a function calls other function that has changed its position because of the update, that call address is not calculated at execution time. Thus, it is necessary to modify the call address of all the functions that use the new function. With this technique it is necessary to erase and rewrite the FLASH memory every time that there is an update, so the execution time and power consumption increases. Besides, the life of the FLASH memory decreases each time that the memory is erased.

Because of the previously commented limitations, in this section an update scheme that combines an incremental update and a dynamic linking is proposed. The proposal is similar to [25] but solves some limitations of that technique. The proposed solution allows a rapid firmware update, a low consumption of energy and minimizes the number of times that is necessary to erase the FLASH memory. This strategy will be explained in the next subsections.

5.3.2 Incremental Update Generation Technique (Transmitter Node)

The difference between the firmware already loaded in the nodes and the new firmware can be minimal. This is quite common due to most of the updates consist of bug fixes or parameters changes [23]. In these cases, the old firmware is very similar to the new firmware. Because of this, it would be important to transmit only the changes between firmware versions and not the whole new firmware. Thus, the differences between the two firmwares can be used to generate the update. As it is shown in Fig. 5.11, the two versions of the firmware are compared and from their difference the update script is generated. In this update script only appears the code that has been modified.

This basic scheme also reduces the packets that have to be transmitted through the network. Unfortunately, although this scheme works very well with minimal changes, such as parameters changes or bugs fixes, with large changes (new functions or new application inclusions), it is necessary to reorder the memory or

Fig. 5.11 Update size reduction using difference method

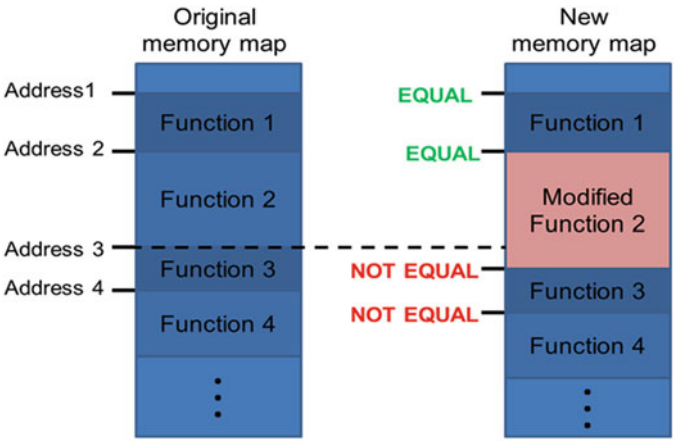


Fig. 5.12 Memory address differences due to a function modification

functions addresses, increasing the time execution and the energy consumption as it was observed in [25]. To alleviate this problem, a technique that performs an efficient ordering of the memory in the receiver node is proposed.

5.3.3 Memory Management for Partial Firmware Update in the Receiver Node

In most of the cases, a small change in the source code results in a very similar firmware image but with different memory addresses or with shifted code. Figure 5.12 shows this problem. It can be observed that a group of functions

(functions 3 and 4) have moved from Address 3 (and so on) to a different address region because new code is inserted. Thus, each call to function 3 and following functions must be performed to a new address. This results in a large update script. In addition, this problem is not limited to functions. There are other objects which memory position may vary such as constant data located in FLASH and global and static variables located in RAM.

It is difficult to obtain efficient updates with the typical memory layout shown in Fig. 5.12. In order to fix this problem, a possible solution is introduced in the next section.

5.3.3.1 Improved Memory Mapping for Partial Firmware Update

The updated functions have to be stored in a free region of the FLASH memory. The proposed methodology will store theses updated functions after the old firmware so the functions that are not updated will maintain their position in the memory. An example of this technique is shown in Fig. 5.13.

The memory map before the updating can be observed on the left side of Fig. 5.13. And the memory map after the updating can be observed in the right side. In this example the updated or modified function is function 1. It is assumed that this updated version was received via air.

This updated function 1 is copied at the beginning of the free space of the FLASH (this space can be allocated at the end of the old firmware as it was previously explained). Because of that none of the non update function addresses are modified. This can be observed on the right side of Fig. 5.13. But still remains the shift problem. How to solve this issue is proposed in the next subsections.

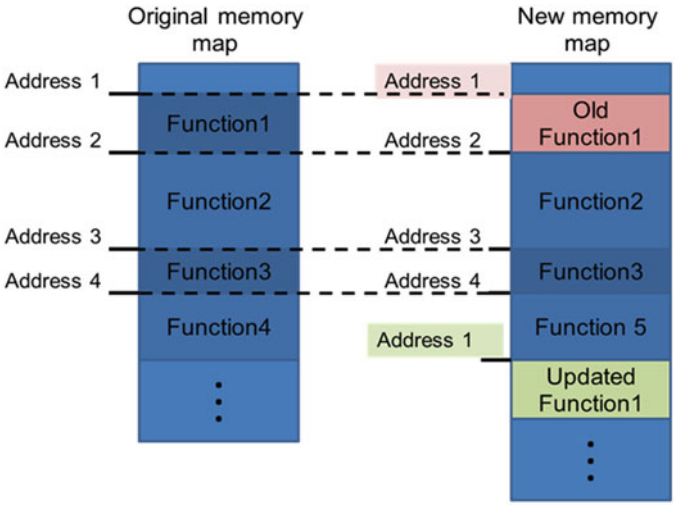


Fig. 5.13 Function mapping proposal

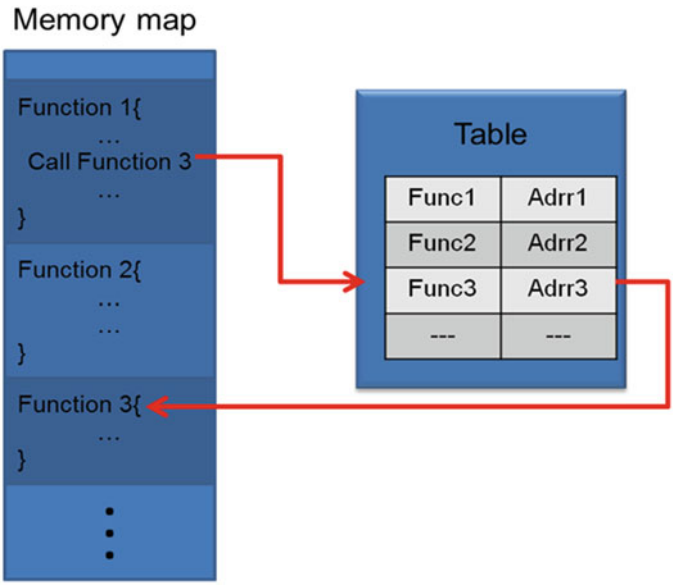


Fig. 5.14 Address table in FLASH

5.3.3.2 The Reference Table Approach

It is possible to avoid the shift problem by using a reference or address table (indirect addressing). For instance, the function calls to function 1 can be performed without problems by accessing to the function address through this address table. Thus, it is possible to point correctly to a function independently of the real position of these functions in memory and it is not necessary to erase the old functions when inserting the update functions.

Figure 5.14 shows an example in which the function 1 needs to jump to function 3 using an intermediate address table. This table stores the address or memory position for each firmware function.

When executing function 1 call, instead of jump to the position of function 3, the program will read to the table position related to function 1 and it will obtain the address of function 3 to finally perform the jump to that address. The process is shown in Fig. 5.13.

In conclusion, it will be not necessary to change the address of the functions that calls to the update function. It will only be necessary to modify the entry of the intermediate table to point to the new address.

5.3.3.3 Improving the Reference Table Approach

When a function is updated and (as it was shown in Fig. 5.12) the position in memory changes, the entry of the intermediate table must be modified with the new

address. Thus, it is still necessary to rewrite the function address. This implies that it would be also necessary to erase its associated section of the FLASH memory. To avoid this, the proposed methodology invalidate the affected table entry and write a new entry in a similar way than when updating functions (as it was shown in Fig. 5.13).

In addition, a second table can be used to improve this process. This second table will be allocated in the RAM memory that has not the FLASH-related problems (it can be rapidly erased and rewritten without increasing too much the power consumption).

To accomplish this, the table of Fig. 5.14 will be copied to the RAM when a new firmware is loaded. The old entries of this table will be eliminated leaving only the valid ones. Figure 5.15 shows this process.

In this case, the update consists in the introduction of a new function (function 2). Because of this update, its position in the memory FLASH has changed (from address 2 to address 4). The old reference is deleted from the address table of the FLASH memory and a new entry at the end of this table is written with the new reference. Thus, only a new line of information has to be added to the FLASH and there is no need to delete the whole block.

Once this is done, the update is complete and the only remaining step is to move this new table to RAM after eliminating the old entries (function 2 in the example) and writing only the update address without varying its position in the table. With this technique the address of each function will be correct even after an update and its position inside the table will not vary.

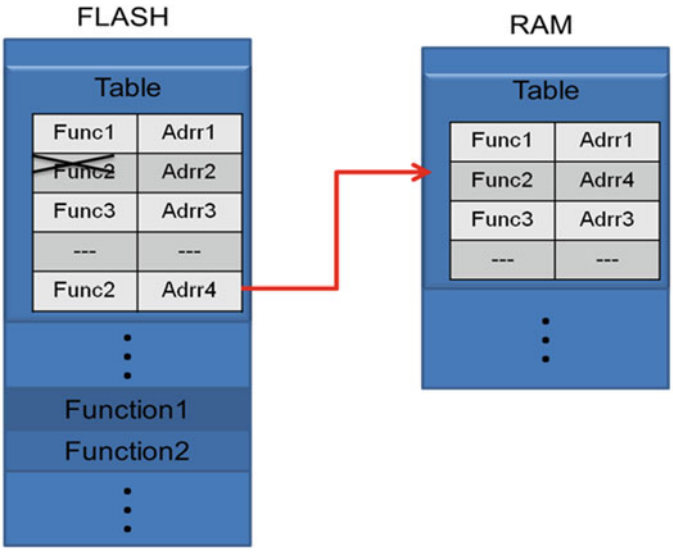


Fig. 5.15 Allocating address table in RAM

5.4 New Security Features in Wireless Sensor Networks

In order to improve protection against attacks in WSN, new security features are needed. This section presents innovative solutions that improve WSN embedded software security. It is important to achieve solutions that do not increase the power consumption. This is a special requirement for these devices that it is necessary to take into account. The proposed solutions include several features such as secure booting, secure data memory, a low power partial update and a new metric for measuring security in virtual platforms oriented to WSN.

5.4.1 Secure Boot

The booting is one of the most vulnerable processes for a wireless device that operates in a non-controlled environment. There are several aspects that need to be checked. In one hand, the loaded firmware is not corrupted or intentionally modified. In the other hand, the loaded firmware is allocated in a secure zone where is not possible to reach for a possible attacker. This section will focus about the integrity and authentication check of the firmware to be loaded and how this can be performed without a significant increase of the power consumption.

One of the first booting-related issues is to choose the memory device where the firmware will be stored. Sometimes the node stores several firmware versions in the same location, thus it is possible to load different firmware releases. Sometimes the storage of a firmware has a memory size that is higher than the internal flash memory of the device. Taking all this into account, the use of an external FLASH memory as storage of the node firmware is highly recommended. Figure 5.16 shows the assumed trusted zone assuming that the target platform is a Silica Xynergy STM32F2 device [33]. The external FLASH is considered insecure, thus a safe booting methodology is needed when the external FLASH memory is used for code storage.

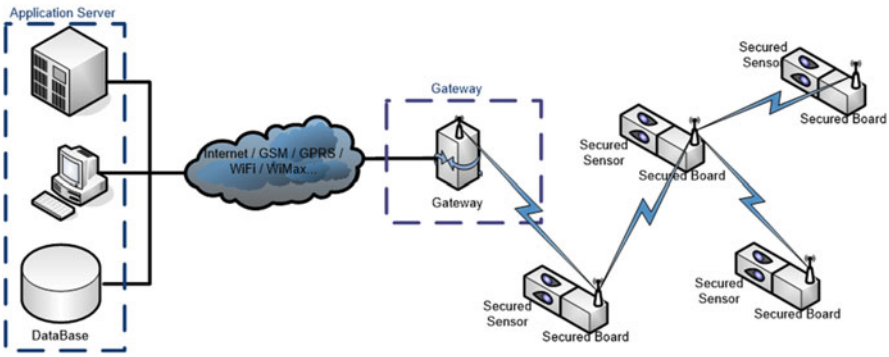


Fig. 5.16 Board trusted and untrusted zone

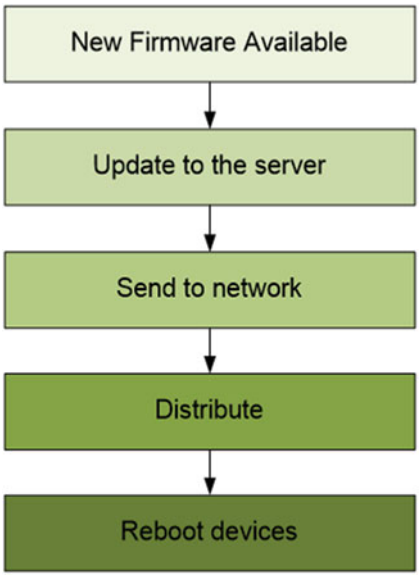


Fig. 5.17 Typical WSN booting

5.4.1.1 Typical Embedded Booting

An example of a generic boot process diagram is showed in Fig. 5.17. This booting process is responsible for accessing the secure block of memory, where the firmware is located, and its extraction. Before the extraction, it is necessary to check the integrity of the firmware. An option to ensure the integrity is to sign the firmware to be loaded. With this solution, it is possible to ensure that the firmware has not been modified or corrupted. After this checking, the firmware-loading process can be executed.

But in this basic booting, confidentiality (that refers to protect the information from disclosure to unauthorized parties) is not accomplished. This could be achieved by encrypting the kernel image that has to be downloaded. It is important to remind that one of the goals when implementing a secure booting in WSN is not to increase the power consumption significantly, thus a careful study of cryptography methods for their implementation in WSN is highly recommended.

Some research works have analyzed the impact of cryptography for WSN. For example, in [47] the theory of trust start-up for an embedded platform is presented. There is a mention to a cryptographic module and key management but there is no specific information about the used method. The importance of measuring power consumption in WSN is pointed in [38] but in this paper the study is only referred to AES methods. In [31], the authors perform a wide analysis on the cost of using symmetric and asymmetric cryptographic algorithms and HASH chain functions but the work is focused on hand-held devices. In [1] there is a comparison between different symmetric cryptographic algorithms. In [9], a novel key management scheme is presented. It is oriented to wireless sensor network security and it is based

on a public key method. This paper focuses on the methodology to create and share the keys for encryption (as RSA or ECC) but there is a lack of practical results confirming the proposed scheme. In [28], authors provide results of experiments with AES and RC4, two symmetric key algorithms that are commonly used in WLANs. However, nowadays the use of RC4 is not so extended as triple DES thus the comparison is not complete. In [40], they present a survey of security issues in WSNs. First, they outline the constraints, security requirements and typical attacks with their corresponding countermeasures in WSNs. Then, they present a holistic view of security issues. One of the conclusions from this paper is that symmetric key cryptography is superior to public key cryptography in terms of speed and low energy cost. However, the key distribution schemes based on symmetric key cryptography are not perfect. Efficient and flexible key distribution schemes need to be designed.

In [24], comprehensive cryptography algorithms suitable for WSN are evaluated. This survey only includes symmetric algorithms as RC4, AES, in a similar way to previous papers. They propose to take into account the factors that may affect algorithm choice, such as clock cycles, code size, SRAM usage, and energy consumption but results are only focused in a few of them.

Most approaches propose a fixed solution for a secure booting, choosing a specific encryption or security procedure, while in this chapter a flexible strategy that allows different level of security/consumption is proposed. This proposal is validated with experimental results that show the power consumption of different cryptography solutions integrated with the booting for a Silica Xinergy STM32F2 device. Similar behavior is expected in other target devices.

5.4.1.2 Cryptography Methods for the Booting

Cryptography methods can be classified into symmetric or asymmetric procedures, depending of the existence of a public key or not. In this chapter, it is proposed the use of symmetric encryption for the booting process because of three important reasons:

- They usually consume less power than asymmetric encryption. This aspect was analyzed in [38, 47].
- Usually the target platform includes specific hardware for symmetric encryption as AES (Advanced Encryption Standard) and triple DES (Data Encryption Standard).
- The private key is deployed on the device during the network initialization phase, simplifying the distribution process and avoiding an increase in the power consumption.

The use of one or several cryptography methods for booting is highly recommended to assure two important things: authentication and confidentiality. Both aspects are important for security but their purpose is different and the methods to accomplish them are also different. There are many different schemes to obtain

authentication or confidentiality in the literature but, as it will be demonstrated with measurements, it is highly recommended to use the cryptography methods directly implemented in hardware since their energy consumption is lower than cryptographic algorithms that are implemented by software. The typical cryptography schemes available in boards are:

- Authentication
  - MD5
  - SHA-1
  - HMAC-MD5
  - HMAC-SHA-1
- Confidentiality
  - TDES
  - AES-128
  - AES-192
  - AES-256

To authenticate the downloaded embedded program, a HASH algorithm can be used and a MAC (Message Authentication Code) signature is obtained. Depending of the desired security level, HASH can be secured by a password (HMAC, HASH-based message authentication code). This signature is usually amended at the end of the firmware image. At the receptor side, the HASH is calculated from the image and it is compared to the HASH included with the firmware. If the matching is correct, the firmware is authenticated. As it happens with the encryption, the target device from this work provides specific hardware for HASH calculation. The functionality of the available HASH hardware include MD5 (Message Digest Algorithm 5) and SHA-1 (Secure HASH Algorithm 1). MD5 creates a 16 bytes HASH while SHA-1 creates a 20 bytes HASH, thus SHA-1 is expected to require a little more power consumption than MD5 while providing a little more security.

To provide integrity and confidentiality the firmware has to be encrypted. It is worth mentioning that AES is a standardized cryptography method that is the successor of DES. It is a symmetric key cryptographic algorithm that efficiently computes the cipher text of a plaintext using a provided key. This efficiency is result of the fact that within the algorithm only bit-operations like XOR or cyclic shifting are applied. And those can be easily implemented in hardware. The specified block size for AES is 128 bits. The keys can have length of 128, 192 or 256 bits. Longer keys provide stronger security guarantees.

5.4.1.3 Consumption Measures of Booting with Security

As it was mentioned, the hardware platform could integrate a cryptography module. In this chapter, a Silica Xinergy Board has been chosen as the target hardware platform. It includes a symmetric cryptography library with the schemes shown in last subsection. If the firmware is encrypted it is necessary to decrypt it during the

booting process. In addition, if a HASH signature of the firmware is included in the firmware, the booting procedure has to calculate the HASH of the received firmware and compare both of them to assure the authentication.

But as it was previously commented it is recommended to know the power consumption of the cryptography methods. In this chapter, real power consumption measures are shown for the Silica Xinergy Board. These measures are specific for this device but similar performance is expected in other embedded devices.

There are some previous works about performance estimation of embedded cryptography. In [31], the analysis concludes that specific hardware usually has better ratio for execution/consumption. However, it is still desirable to run some tests in the device to verify these results for the target platform.

For encryption and decryption tasks, the micro-controller STM32F217 of the Silica Xinergy Board has specific hardware modules that implements different cryptographic algorithms as TDES, AES-128, AES-192 and AES-256. The term AES-"KeySize" is related to the size of the used key. The different algorithms have been executed in hardware and software to compare the execution time and the power consumption.

Related to its performance, it was observed in [1] that AES-128 is more secure than 3DES and, SHA-1 is more resistant to collisions (probability of obtaining the same HASH for two different images) than MD5. In Fig. 5.18, it is shown a security comparison (obtained from [8, 9]) of different encryption methods.

The tested encryption algorithms are the AES with different key sizes and triple DES. Figure 5.19 shows the execution time and power consumption for the different algorithms in the hardware platform. The testbench consists of the encryption of a 25,600-bytes text with a clock frequency of 120 MHz.

The AES hardware implementation is around 10× faster than the software implementation, and the power consumption has the same ratio. In the case of Triple DES, the software implementation is slower and the ratio is bigger. Specifically for the software case, the consumption of TDES is 7× higher than AES-128 and 5.4× higher than AES-256. For the hardware case, the consumption of TDES is 3× higher than AES-128 and 2.2× higher than AES-256. For the decryption algorithm, the results are very similar.

For authentication, the tested algorithms were the SHA-1, MD5, HMAC SHA-1 and HMAC MD5. Results for these methods are shown in Fig. 5.20. The decrypted text is the same that the one used to test the encryption algorithms. The ratio between software and hardware execution time is variable. The MD5 algorithm in software is only 4× worst than hardware. The consumption of SHA-1 in software is 17× higher than in hardware. The HMAC versions show different ratios. Whereas the HMAC MD5 in software has a higher consumption of 3× compared to the hardware case, the consumption of HMAC SHA-1 in software is 20× worst than in hardware.

It is also observed that although the software implementation of SHA-1 requires more power than the software implementation of AES but in the hardware case is AES who requires more power than SHA-1.

According to these results some methods can be discarded and other can be chosen as optimum for a secure booting. The chosen ones in this proposal are HMAC-SHA1 (for authentication), AES-128 and AES-256 (for encryption).

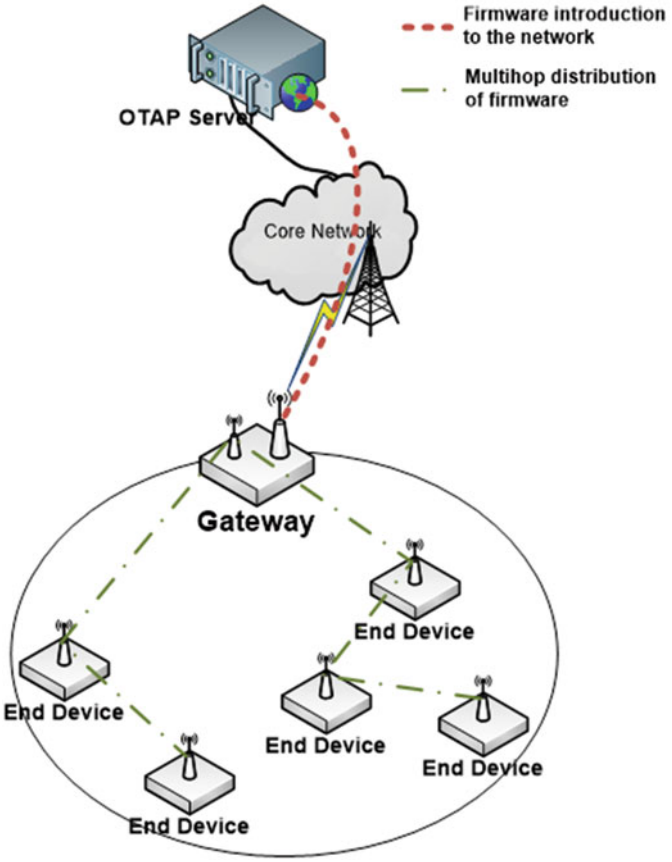


Fig. 5.18 Security for the different methods [8, 9]

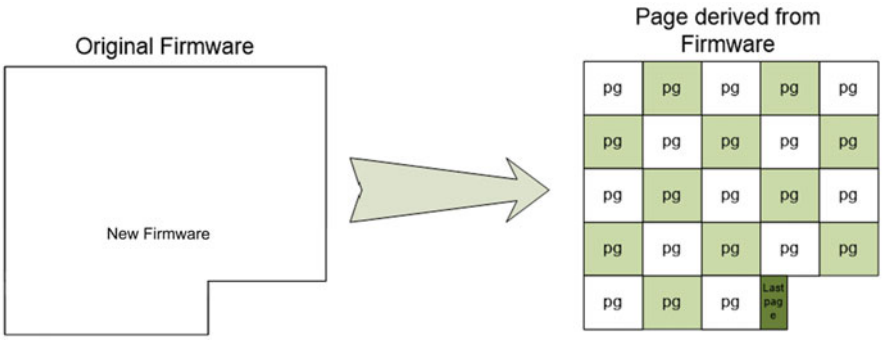


Fig. 5.19 Consumption of symmetric encryption algorithms

Fig. 5.20 Consumption of HASH algorithms

Fig. 5.21 Example of energy measurement

But still is necessary to measure these methods integrated with the designed booting process. In order to do that, a direct measurement of the board power consumption was performed. In this case the booting process de-encrypts the encrypted firmware using an AES-128. The parameter of the x-axis is time in seconds and the values of the y-axis is voltage. In this snapshot of the power consumption behavior of the system, different working zones are identified. The total consumed energy is calculated by multiplying the power by the total time that the booting is being executed (Fig. 5.21).

Figure 5.22 shows the measures obtained of the cryptography methods integrated in the booting process. In this figure is shown the energy consumption increase compared to a booting without security. It is also shown the difference between using hardware of software methods, thus a more clear vision about the impact of this choice would have in the booting.

First, it can be observed that software methods extremely increase the energy consumption. The software implementation of the most secure cryptographic combination (HMAC-SHA-1 + AES-256) almost increased a 100 % the energy consumption in the booting compared with the hardware implementation. Thus, software methods should be discarded to accomplish security in the booting. It is also observed that the authentication performed with HMAC-SHA-1 consumes more than AES-128 encryption. This result confirms other results that have been described in the literature.

5.4.1.4 Flexible Booting Procedure with Different Security Levels

The typical booting process previously described does not include confidentiality. However, it is possible to add this feature and transform it in a more secure booting. In this chapter, instead of a fixed secure booting scheme, a flexible procedure is proposed. The idea is that not for every node it is necessary to perform a secure

Fig. 5.22 Percentage of increased of power consumption

booting, since it depends of its purpose, deployment zone, etc. So, in order to alleviate the power consumption, it is possible to choose what level of security (that has associated a level of power consumption) is desired. So, it will be possible to choose no security at all, to choose only authentication to be performed, confidentiality or both.

This methodology will take into account the necessary security to prevent attacks while maintaining a low consumption. In fact, it can be tuned depending of the expecting risk and/or purpose of the deployed nodes. The application software has to decide how much power consumption overhead it can bear to get a higher security. It is a trade-off decision left open to the application. This method outperforms most

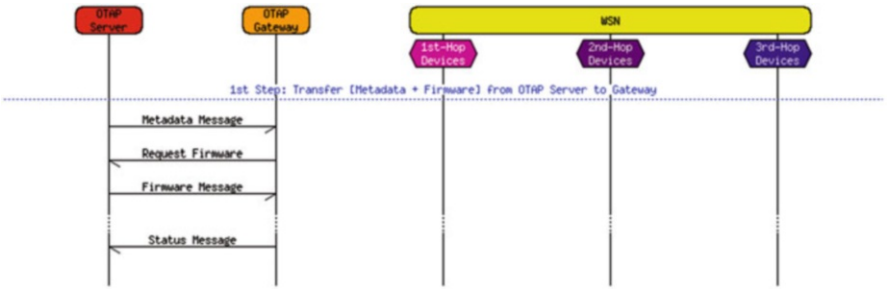


Fig. 5.23 Power consumption associated for each level

of the proposal in the literature in security, flexibility and power consumption. The proposed security levels were chose according to the results shown in Fig. 5.22 and they are:

- **Level 0. No encryption nor authentication**
- **Level 1. Creation of a secure signature with HMAC-SHA-1**
- **Level 2. Firmware encryption with AES-256**
- **Level 3. HMAC-SHA-1 + firmware encryption with AES-256**

These security levels are ordered from less secure to more secure. Of course, this increase in security has associated an increase in power consumption. It can be observed in Fig. 5.23 the power consumption associated for each security levels. The designer can decide which one is the most appropriate.

It is worth to mention since AES-256 almost consumes the same energy (less that a 2 % of difference) than AES-128, it was preferable to implement only AES-256 because of its higher security compared to AES-128.

Although, this proposal is focused for the specific device commented in this chapter, similar results are expected for other devices. In addition, it is possible to replicate the methodology explained in this section in other boards if the developers want to assure these results and conclusions.

5.4.2 Secure Virtual Environment

Once the operating system and the main application are loaded in the memory using a secure boot, it is possible to assure that the loaded firmware is not corrupted and/or modified. However, it is still necessary to secure the memory where the OS and the application are stored. Since the OS is one of the most typical “entry points” for an attack, the modification or update of an OS kernel is critical for the systems security. Typical limitations of WSN embedded software can simplify attacks in some aspects, since operating systems for sensors usually do not use memory

protection and do not distinguish kernel mode or user mode execution though its attack is challenging because the limited size of a single wireless message imposes a hard limit into size of malicious code.

There are different ways to corrupt the OS when it is loaded and executed in the system. One of the most typical attacks is based on the load of a corrupted application that accesses directly to the OS data or code area. This causes an important problem because when an application has access to the OS, its alteration is relatively simple. Even an erroneous application developed by the network owner may cause a security hole (Fig. 5.24). Because of this, the OS must be loaded in a secure memory portion (kernel space) where the applications cannot access it directly. Two approaches to try to solve this problem are explained in this subsection of the chapter. One focuses in the use of the Hardware Memory Protection Unit that can be available in some of the devices that compose the WSN nodes. The other approach relates to an updating philosophy in which the OS or the applications are updated independently.

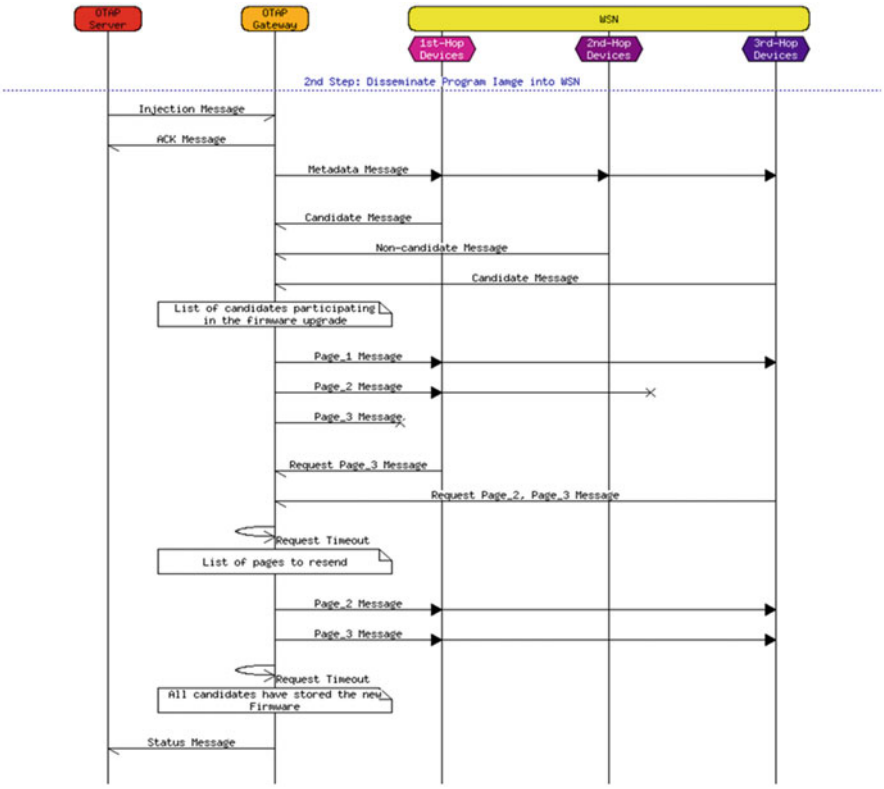


Fig. 5.24 Example of a security hole

5.4.2.1 Use of the Memory Protection Unit

The Hardware Memory Protection Unit (MPU) is integrated in the target device STM32F2 that is used as example device for a node in this chapter but is available in most devices available in the market.

This MPU provides support for protecting regions of memory through enforcing privilege and access rules. It supports up to eight different regions, each of which can be split into a further eight equal-size sub-regions. These sub-regions are useful for overlapping memory regions. For example, if you have a large region but would like different attributes for a small section of it. A sub-region in the larger region could be disabled and a second MPU region used for that sub-region to provide the required attributes. The MPU can also protect system peripherals from unintended modification by tasks. Other applications would require the isolation of un-trusted code and a guarantee of detecting task stack overflows. The required MPU basic operation consists of these steps:

1. Protect a section of RAM
2. Access the RAM
3. Run the exception routine if access is denied

In summary, the objective is to create an independent secure kernel space where applications do not have direct access. This allows storing the kernel in a safe memory zone, thus preventing attacks that could maliciously modify the OS through the application. Moreover, the Memory Protection Unit will protect the OS and the authenticated applications from a number of potential errors as system or hardware failures. Figure 5.25 shows the proposed memory scheme. It can be appreciated that there are three main sections in this figure. The first section is the operating system table. The second one is reserved for the operating system protected by the MPU, and the third section is used to store the applications.

So, the MPU can be used to protect regions of memory and if there is any attempted, unauthorized access to certain memory regions, then a memory protection violation exception will occur and the system will detect the illegal access.

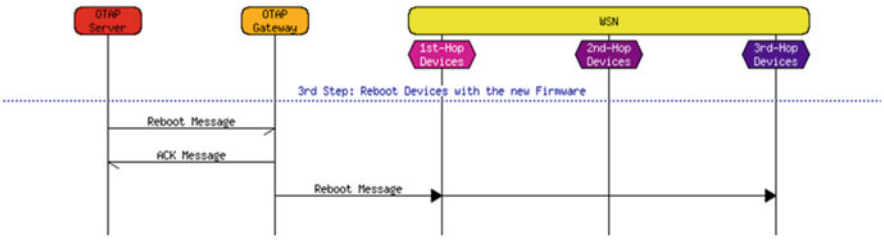


Fig. 5.25 Memory scheme

5.5 Implementing Key Management in the FPGA

The security issues for the key storage and key distribution in the wireless sensor meshed network are described in this subsection. Here the reader can find the use given to the FPGA of the application, the designed IP cores for the secure interface between the ARM and the FPGA and the IP designed to perform the key management inside the FPGA.

5.5.1 Describing the Selected Algorithm AES-256

As it was mentioned before in prior sections, the system will be protected by a symmetric encryption algorithm that will cipher the data that are going to flow through the network. (See [4] for a survey on types of encryption.)

The selected encryption algorithm to perform that operation is AES-256 [5, 44], this encryption algorithm is based on symmetric encryption, its acronym means Advanced Encryption Standard and is an stronger encryption standard, which uses the Rijndael algorithm. This algorithm was developed by Joan Daemen and Vincent Rijmen. AES is the successor of Data Encryption Standard XORed (DESX) and 3DES and has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. AES operates on a 4×4 column-major order matrix of bytes AES and the key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plain-text, into the final output, called the cipher-text. The number of cycles of repetition are as follows:

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys.

According to the analysis presented in Sect. 5.4, the selected key for the crypto-system is 256-bit key, as the purpose is to maximize the security of the mesh, and it is applied to encrypt the data and also to reverse the process, to decrypt.

Summarizing as much as possible to simplify the process, the algorithm starts with a random number as a seed, in which the key and data encrypted with it are scrambled applying four rounds of mathematical processes. The four rounds are described next:

1. SubBytes: A look-up table is used to determine what each byte is replaced with.
2. ShiftRows: Has a certain number of rows where each row of the state is shifted cyclically by a particular offset, while leaving the first row unchanged. Each byte of the second row is shifted to the left, by an offset of one, each byte in the third row by an offset of two, and the fourth row by an offset of three. This shifting is applied to all three key lengths, though there is a variance for the 256-bit block where the first row is unchanged, the second row offset by one, the third by three, and the fourth by four.

3. MixColumns: This step is a mixing operation using an invertible linear transformation in order to combine the 4 bytes in each column. The 4 bytes are taken as input and generated as output.
4. AddRoundKey: Derives round keys from Rijndael's key schedule, and adds the round key to each byte of the state. Each round key gets added by combining each byte of the state with the corresponding byte from the round key.

Lastly, these steps are repeated again for a fifth round, but do not include the MixColumns step.

The FPGA in this application case is essential to implement in a safe manner the mechanism that can do the tasks that control the key generation [26], key predistribution, key deployment and key storage of AES-256 [6]. That is why the FPGA in the device (mote or gateway) is going to be employed on that tasks, as it is pure HW and harder to violate. All that mechanisms have to be prepared to produce and serve 2 type of keys:

- The external keys: Will be the keys that will secure the data (firmware or application) and also will be used to protect the new delivered external keys.
- The internal keys: Used inside the device (mote or gateway) and unique in each device to protect the external keys, Application or OTAP. This key will be generated with the purpose of encrypting the data that is going to flow in the FSMC bus. This data will be encrypted with AES-256 algorithm too and it will be generated stored and renewed if necessary by the FPGA in the GATE or Mote.

Let's focus a bit on the external keys that must be provided and their uses. As it has been mentioned, 2 keys are needed to assure the LPWSN. The first one is the OTAP key, this is the key for the Over-The-Air-Programming protocol implemented. As it was explained before, the basis for the OTAP is to distribute firmware split in packets with the purpose of being downloaded in certain nodes of the LPWSN. Then this pieces will be assembled composing the entire firmware image that will be installed in the node. After the download and installation, a reboot is necessary to run the new firmware version. The system will do this taking into account the devices capabilities, consumption and communication overhead. It consist of three parts:

1. The OTAP server: There is defined the firmware version to be distributed and manages the OTAP processes
2. The gateway: Who will be the only element connecting the OTAP server with the end devices.
3. End Device: Is the final element that will receive the firmware to be installed.

The transmission of packets OTAP server-Gateway (through wireless channel) and Gateway-End Device (through Wireless too) must be encrypted, this will cipher the binary code preventing it to be read by attackers [4]. So a key for that will be needed and the FPGA will be in charge of generating and renewing it with the mechanisms that will be described further on in this document.

The second key that the system will need is the Application key, this key will be secured too, as it is transferred from the FPGA to the microprocessor through

the FSMC channel and therefore can be susceptible to some attack [4]. This key will secure the captured data by the sensors in the motes and will be useful for the application operation.

5.5.1.1 The Secure Communication Channels

The keys (OTAP or Application) have to be communicated outside the FPGA, which means it has to go out the secure place, and have to travel in the different media such as the communication channel between the microprocessor and the FPGA or through the Xbee among the Gateway and the Motes, which are considered unsafe zones. Although the key will not be transferred on plain text but in ciphered text, to secure the communication channel a CRC has been implemented.

Cyclic Redundancy Check (CRC)

The Cyclic Redundancy Check is an error-detecting code with an extended usage in digital networks and storage devices to detect changes made to raw data. The blocks of data enter to this system and a value is attached to them basing it on the remainder of a polynomial division of their contents.

The CRCs added is redundant, that means that it expands the message without adding information, but it allows to be authenticated. CRCs are commonly used because they are simple to implement, easy to analyze, and good at detecting common errors caused by noise in transmission channels. Because the check value has a fixed length, the function that generates it is occasionally used as a hash function. From its invention in 1961 by W. Wesley Peterson[27], the 32-bit CRC function of Ethernet and many other standards had been very used [36].

Based on that, at both extremes of communication CRC-32 has been implemented, similar approaches can be seen in [2,46]. The difference in this authentication with the common implementations in the field is that the CRC is encrypted: The CRC is added to the data to be transmitted and then encrypted with the encryption key. In this way, the receiver can only check the CRC if it has the key to decode, avoiding a possible attack with the purpose of violating the CRC code. In Fig. 5.26 it can be seen the authentication sequence, preparing the message to be delivered outside the FPGA.

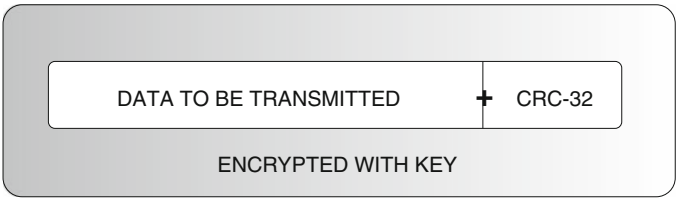


Fig. 5.26 The CRC addition to the data to be transmitted

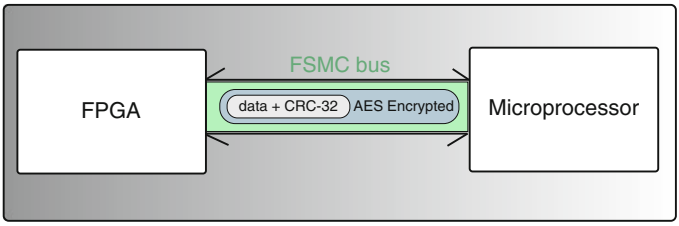


Fig. 5.27 Securing and authenticating the communication channel inside the device (FSMC)

Communication *Intra-device* Through Flexible Static Memory Controller (FSMC) Bus

As it was mentioned before, there will be two communication channels in which the data are going to be sent. One of them is the FSMC (see [37] for further details on the bus controller), that is the parallel bus which is connecting the ARM and the FPGA. This bus consist of 16 data lines, 26 address lines and 6 control signals. Through them, there will be sent the data encrypted together with the CRC and in such a way it allows to assure and authenticate the channel in front of any physical attack. Through the FSMC bus, the external and internal keys are going to be shared between ARM and FPGA in all of their possible states: initial, pre-distributed, distributed or renewed keys, always authenticated via CRC-32 and encrypted. See Fig. 5.27.

Wireless Communication *Inter-device*

The other communication channel is the RF signal, Xbee, sent from the Gateway to the Motes involved in the network. The data that are going to be transmitted here are of two types: data packets (from firmware or application) and external keys (OTAP and Application), both will be authenticated with a CRC for a later encryption with the OTAP current key. See Fig. 5.28 for an schematic approach. In this case, the data received is decoded by the microprocessor and the CRC it's checked by it too.

5.5.2 Key Management in the FPGA

As previously mentioned, the FPGA role will be the key management of the crypto-system. This task is extremely important, because any mistake in the key management may lead in the discovery of the key by an attacker, making all the network vulnerable and its generated data too. A similar approach on symmetric key management with secure storage is described in [15].

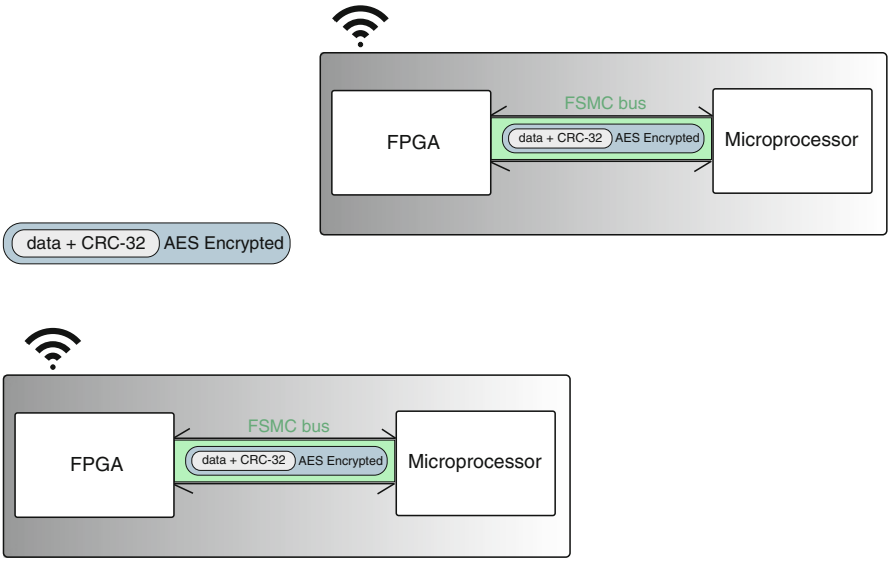


Fig. 5.28 Securing and authenticating the wireless communication channel inter-device

The basic modules developed inside the FPGA as key management functions or IPs are detailed in the next subsections:

5.5.2.1 Key Generation

Both internal key and external keys must be generated inside the FPGA to assure its privacy, and for that they are going to be randomly formed. The key generation process is an IP (Intellectual Property) inside the FPGA that delivers 256 bit key pseudo-random generated. This pseudo-random generation is based on polynomials (see Table 5.1 for some examples of polynomials that may conform the Linear Feedback Shift Registers that operate inside it). From this logic combination two keys for encryption are going to be delivered. These key will allow to encrypt the distributed firmware and the data sent. In Fig. 5.29 the insides of the IP for the keys generation are shown.

5.5.2.2 Pre-distribution

Before the keys' creation process, both sides of the communication (Microprocessor-FPGA or MOTE-GATE) must know the initial key with which the system starts encrypting the data. When requested, a module in the FPGA will be in charge of making the pre-distribution. The initial key is created for that purpose, so the

Table 5.1 Some examples on possible polynomials to be used [42]

n	XNOR from	n	XNOR from	n	XNOR from	n	XNOR from
3	3,2	45	45,44,42,41	87	87,74	129	129,124
4	4,3	46	46,45,26,25	88	88,87,17,16	130	130,127
5	5,3	47	47,42	89	89,51	131	131,130,84,83
6	6,5	48	48,47,21,20	90	90,89,72,71	132	132,103
7	7,6	49	49,40	91	91,90,8,7	133	133,132,82,81
8	8,6,5,4	50	50,49,24,23	92	92,91,80,79	134	134,77
9	9,5	51	51,50,36,35	93	93,91	135	135,124
10	10,7	52	52,49	94	94,73	136	136,135,11,10
11	11,9	53	53,52,38,37	95	95,84	137	137,116
12	12,6,4,1	54	54,53,18,17	96	96,94,49,47	138	138,137,131,130
13	13,4,3,1	55	55,31	97	97,91	139	139,136,134,131
14	14,5,3,1	56	56,55,35,34	98	98,87	140	140,111
15	15,14	57	57,50	99	99,97,54,52	141	141,140,110,109
16	16,15,13,4	58	58,39	100	100,63	142	142,121
17	17,14	59	59,58,38,37	101	101,100,95,94	143	143,142,123,122
18	18,11	60	60,59	102	102,101,36,35	144	144,143,75,74
19	19,6,2,1	61	61,60,46,45	103	103,94	145	145,93
20	20,17	62	62,61,6,5	104	104,103,94,93	146	146,145,87,86
21	21,19	63	63,62	105	105,89	147	147,146,110,109
22	22,21	64	64,63,61,60	106	106,91	148	148,121
23	23,18	65	65,47	107	107,105,44,42	149	149,148,40,39
24	24,23,22,17	66	66,65,57,56	108	108,77	150	150,97
25	25,22	67	67,66,58,57	109	109,108,103,102	151	151,148
26	26,6,2,1	68	68,59	110	110,109,98,97	152	152,151,87,86
27	27,5,2,1	69	69,67,42,40	111	111,101	153	153,152
28	28,25	70	70,69,55,54	112	112,110,69,67	154	154,152,27,25
29	29,27	71	71,65	113	113,104	155	155,154,124,123
30	30,6,4,1	72	72,66,25,19	114	114,113,33,32	156	156,155,41,40
31	31,28	73	73,48	115	115,114,101,100	157	157,156,131,130
32	32,22,2,1	74	74,73,59,58	116	116,115,46,45	158	158,157,132,131
33	33,20	75	75,74,65,64	117	117,115,99,97	159	159,128
34	34,27,2,1	76	76,75,41,40	118	118,85	160	160,159,142,141
35	35,33	77	77,76,47,46	119	119,111	161	161,143
36	36,25	78	78,77,59,58	120	120,113,9,2	162	162,161,75,74
37	37,5,4,3,2,1	79	79,70	121	121,103	163	163,162,104,103
38	38,6,5,1	80	80,79,43,42	122	122,121,63,62	164	164,163,151,150
39	39,35	81	81,77	123	123,121	165	165,164,135,134
40	40,38,21,19	82	82,79,47,44	124	124,87	166	166,165,128,127
41	41,38	83	83,82,38,37	125	125,124,18,17	167	167,161
42	42,41,20,19	84	84,71	126	126,125,90,89	168	168,166,153,151
43	43,42,38,37	85	85,84,58,57	127	127,126		
44	44,43,18,17	86	86,85,74,73	128	128,126,101,99		

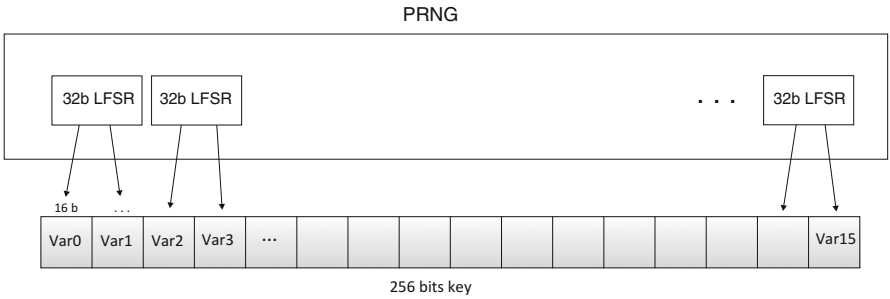


Fig. 5.29 IP internal scheme for keys generation

process of the pre-distribution key is in charge of delivering the initial key to both sides of the channel. The pre-distribution is done in the booting process of the system. See [45] for other possible pre-distribution approaches.

5.5.2.3 Deployment

The process of deployment is going to be necessary after the process of pre-distribution because it assures the anonymity and privacy of the given keys. This process will use the IP for the key generation inside the FPGA, making this IP to produce the necessary keys to have the secret keys in the system for the first time and in further renewals.

In Fig. 5.30 the necessary steps for the pre-distribution key that will assure the security inside the device are shown. As it can be seen some ACKs messages are needed in order to confirm the pre-distribution process.

Figure 5.31 shows a similar procedure to deploy the AES-256 key that is going to rule the network from that moment on. Note that in this case, the new key is going to be distributed through the wireless and the same communication procedure is going to be used whenever a renewal of the AES-256 key is needed.

The next subsections work as a whole inside the FPGA because they need each other, but for a better understanding of the reader are split into two sections:

5.5.2.4 Key Renewal

As the keys can not be always the same keys (because this will make the system vulnerable to attackers), they must be renewed following certain criteria depending on the security requirements of the system. They can be renewed by many other circumstances, but for the use case application discussed in this chapter, three options have been prioritized:

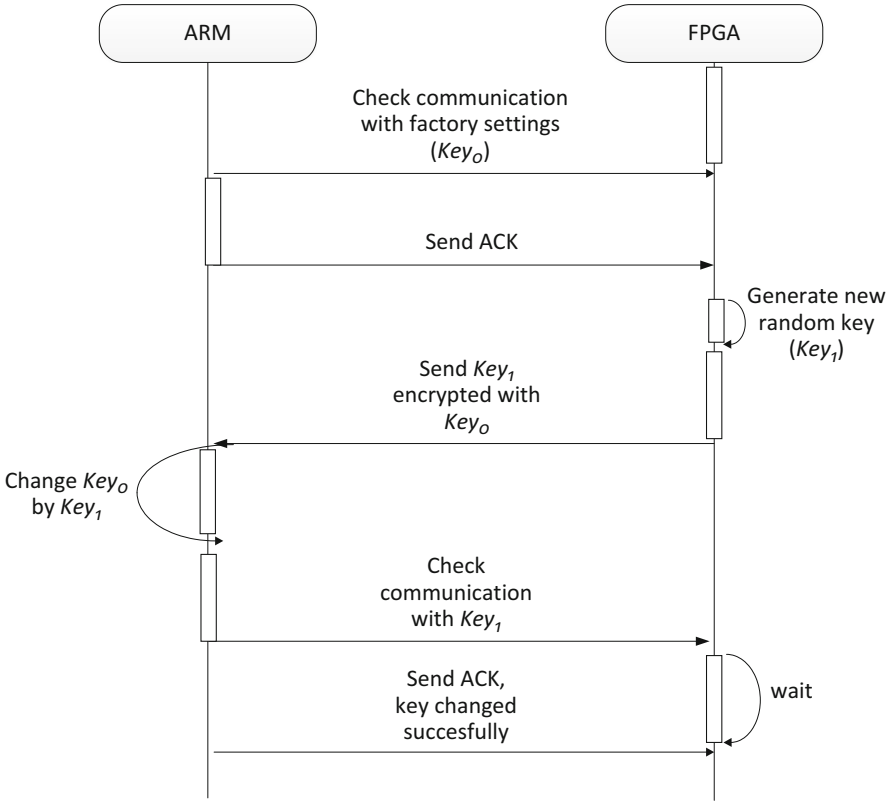


Fig. 5.30 Pre-distribution of the internal key between the microprocessor and the FPGA on the device (Mote/Gateway)

1. By time: Each certain programmed time, the gateway will generate a new key and deliver it to the rest of the system, making all the system change to the new encryption key. This time must be short enough to guarantee that any system can discover the encryption key.
2. By alarm received from the end devices (motes): If the device is being attacked or manipulated, the accelerometer will detect the manipulation and launch an alarm signal to the Gateway that will make the system to renew the keys.
3. By request: If a device (mote) requests the renewal of its keys to the Gateway, the Gateway will deliver the new keys to the system.

5.5.2.5 Key Storage

Linked with the prior module is the key storage module. In order to keep the keys secret and in a safe place, the FPGA has been used as an internal storage. Moreover, the key will not be stored in plain text but encrypted with an internal key that will

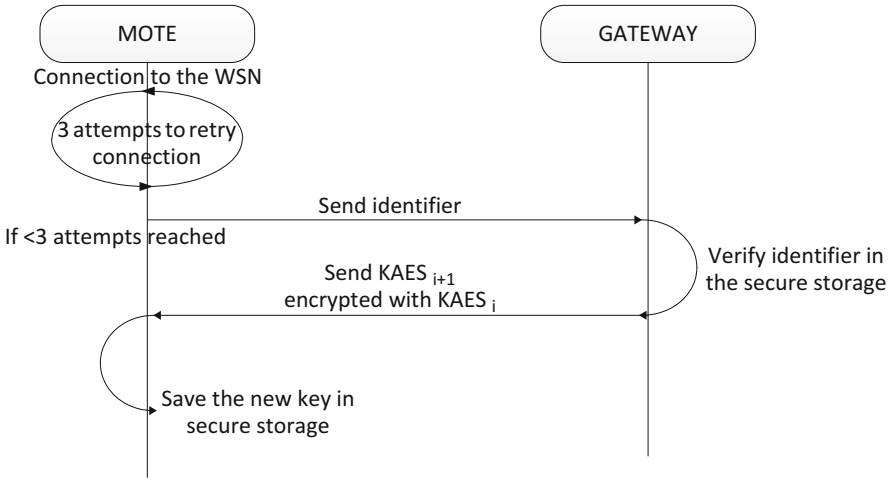


Fig. 5.31 Pre-distribution of the wireless key between the Gateway and the Motes on the LPWSN

protect them. So the renewal module will produce a key each time it is asked to do so, and will keep the delivered key encrypted inside it. As the system asks for a new key, the old one is replaced by the encrypted new one.

5.5.3 FPGA Implementation Possibilities

The main advantage of using the FPGA in the design of the system is that it allows to implement diverse algorithms depending on the level of security required for the application. It is especially relevant as WSNs are being used in a wide range of applications with increasing complexity. This diversity demands of the components of the network the possibility to be updated or adapted to changing conditions that may not be specified when designing the system.

It could happen that the precise requirements will not be available or could not be predicted for the complete lifetime of the network at the moment of designing the nodes. To overcome with these limitations FPGA devices have been used [41]. Due to the nature of this reconfigurable hardware, it is possible to change the implemented hardware systems to adapt them to new requirements. This characteristic allows to modify the part of the node configured in the FPGA, and by doing this, increasing the flexibility of the WSN.

The initial approach for this use case considered the FPGA as a co-processor to implement both the cryptographic algorithm and the key management. It was required as the STM32F217 Micro-controller does not include any module for such tasks. With the appearing of recent families of this Micro-controller (STM32F417), a dedicated module is available to perform the cryptographic tasks. Besides,

the STM32F417 presents better power efficiency compared to the FPGA and for that, the current application performs the tasks of the AES-256 algorithm using this Micro-controller [32].
Despite this, the FPGA is also prepared with an AES-256 cryptographic module to guarantee a strong key management by protecting the FSMC channel with a robust algorithm. Depending on the specified level of security, the FPGA can implement diverse actions. From simple and fast operations such as XOR, to symmetric, asymmetric, hybrid or even custom encryption algorithms, that could be specially designed for the application field.

Although FPGAs are less efficient in terms of power consumption compared to Micro-controllers [10], the possibilities these devices offer in terms of flexibility and performance make them especially suitable for high computing demanding applications such as video processing, real time remote sensing, streaming, etc. Moreover, not all the available algorithms used in cryptographic systems can be performed in the Micro-controller respecting the performance constraints. If a cryptographic algorithm is not supported by the dedicated module of the Micro-controller, the execution time of the corresponding implementation in software could not meet the timing requirements. This, as a result of the increased number of operations executed that can even rise the power consumption. In these cases the FPGA can act as the appropriate alternative to satisfy the requirements of the system.

The hardware acceleration performed by the FPGA can be implemented as complete algorithms such as a cryptographic standard, or smaller parts such as multipliers or matrix operations. The design of such accelerators can follow diverse strategies [11].

One common method is to generate the FPGA configuration file with all the possible algorithms that the application will require. This approach faces the limitations of the silicon area available in the device and the power efficiency. Normally only one algorithm is used at a given time, therefore, the rest of the FPGA will be using resources and consuming power in an inefficient way.

Another approach is to design every algorithm independently and for each of them to generate a configuration file to program the FPGA. These configuration files should be stored in flash memory and the application decides what algorithm to use and launch the reconfiguration of the FPGA. Compared with the first approach, this one has the advantage of use more efficiently the area of the chip. By doing this, smaller devices with the corresponding lower power consumption can be used. The main disadvantage of this method is that the time required for the full reconfiguration of the chip can be considerable for the application and the storage of the configuration files demands a flash memory with enough capacity.

To overcome the last limitations, the recent advances in the field of FPGAs have brought Dynamic Partial Reconfiguration [16]. By using this, it is possible to reconfigure certain parts of the device without affecting the remaining part of the FPGA. This characteristic is specially suitable for WSNs where the power efficiency of FPGAs can limit its utilization. DPR allows the utilization of smaller devices to implement the same algorithms that in a static implementation requires larger chips.

Instead of implementing a complete algorithm, it can be split into diverse tasks. For each task a partial configuration file is generated and stored in a flash memory. Depending on the requirements of the application, each task can be dynamically configured. With this approach the reconfiguration time, the sizes of the files to reconfigure specific parts of the chip and the power consumption can be reduced.

In conclusion, the FPGA could play an important role in future implementations of WSNs with heterogeneous components in which diverse devices could interact among them. As such devices could be able to communicate with different gateways, nodes, servers, etc. The FPGA could tremendously help in adapting cryptographic algorithms or protocols. This flexibility will allow a correct and secure communication. Depending on the constrained-device protocol, the FPGA-based platform will configure the required security algorithm and protocol to allow the communication with new devices. Instead of fixed implementations of all possible security protocols, dynamic partial reconfiguration of the desired protocol will be better in terms of area utilization and power consumption.

References

1. Alanazi, H.O., Zaidan, B.B., Zaidan, A.A., Jalab, H.A., Shabbir, M., Al-Nabhani, Y.: New comparative study between DES, 3DES and AES within nine factors. Journal of Computing. CoRR **abs/1003.4085**, South Korea **2**(3) (2010). ISSN 2151–9617
2. Anton, C., Ionescu, L., Tutanescu, I., Mazare, A., Serban, G.: FPGA-implemented CRC algorithm. In: Applied Electronics, 2009 (AE 2009), Plisen, pp. 25–29 (2009)
3. Balani, R., Han, C.C., Rengaswamy, R.K., Tsigkogiannis, I., Srivastava, M.: Multi-level software reconfiguration for sensor networks. In: Proceedings of the 6th ACM & Amp; IEEE International Conference on Embedded Software (EMSOFT '06), Seoul, pp. 112–121. ACM, New York (2006). doi:10.1145/1176887.1176904. http://doi.acm.org/10.1145/1176887.1176904
4. Chen, X., Makki, K., Yen, K., Pissinou, N.: Sensor network security: a survey. IEEE Commun. Surv. Tutor. **11**(2), 52–73 (2009). doi:10.1109/SURV.2009.090205
5. Daemen, J., Rijmen, V.: The Rinjdael block cipher, AES proposal available via DIALOG (1999). http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf
6. Deshpande, A., Deshpande, M., Kayatanavar, D.: FPGA implementation of AES encryption and decryption. In: 2009 International Conference on Control, Automation, Communication and Energy Conservation (INCACEC 2009), Erode, India, pp. 1–6 (2009)
7. Dunkels, A., Finne, N., Eriksson, J., Voigt, T.: Run-time dynamic linking for reprogramming wireless sensor networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06), Boulder, USA, pp. 15–28. ACM, New York (2006). doi:10.1145/1182807.1182810. http://doi.acm.org/10.1145/1182807.1182810
8. EE Times: http://www.eetimes.com/design/embedded-internet-design/4372428/How-secure-is-AES-against-brute-force-attacks. July 5 (2012)
9. Eldefrawy, M.H., Khan, M.K., Alghathbar, A.: A key agreement algorithm with rekeying for wireless sensor networks using public key cryptography. In: International Conference on Anti-Counterfeiting Security and Identification in Communication, Chengdu, China (2010)
10. Elkateeb, A.: RH-mote for next-generation wireless sensor networks. Procedia Comput. Sci. **21**(0), 217–224 (2013). doi:http://dx.doi.org/10.1016/j.procs.2013.09.029. http://www.sciencedirect.com/science/article/pii/S1877050913008223. The 4th International Conference

on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013) and the 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH)
11. Gonzalez, I., Lopez-Buedo, S., Gomez-Arribas, F.J.: Implementation of secure applications in self-reconfigurable systems. Microprocess. Microsyst. **32**(1), 23–32 (2008). doi:10.1016/j.micpro.2007.04.001. http://dx.doi.org/10.1016/j.micpro.2007.04.001
12. Han, C.C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A dynamic operating system for sensor nodes. In: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys '05), Seattle, USA, pp. 163–176. ACM, New York (2005). doi:10.1145/1067170.1067188. http://doi.acm.org/10.1145/1067170.1067188
13. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04), Baltimore, USA, pp. 81–94. ACM, New York (2004). doi:10.1145/1031495.1031506. http://doi.acm.org/10.1145/1031495.1031506
14. Jeong, J., Culler, D.: Incremental network programming for wireless sensors. In: 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (IEEE SECON 2004), Santa Clara, USA, pp. 25–33 (2004). doi:10.1109/SAHCN.2004.1381899
15. Kazienko, J., Albuquerque, C.: Secure secret key distribution and storage in wireless sensor networks. In: 2010 IEEE 10th International Conference on Computer and Information Technology (CIT), Bradford, United Kingdom, pp. 890–895 (2010). doi:10.1109/CIT.2010.166
16. Koch, D.: Partial reconfiguration on FPGAs, architectures, tools and applications. Lecture Notes in Electrical Engineering 153, Springer, pp. 1–252 (2013), ISBN 978-1-4614-1224-3
17. Koshy, J., Pandey, R.: Remote incremental linking for energy-efficient reprogramming of sensor networks. In: Proceeedings of the Second European Workshop on Wireless Sensor Networks, Istanbul, Turkey, pp. 354–365 (2005). doi:10.1109/EWSN.2005.1462027
18. Koshy, J., Pandey, R.: VM*: synthesizing scalable runtime environments for sensor networks. In: Proceedings of the Third International Conference on Embedded Networked Sensor Systems (Sensys), San Diego, USA, pp. 243–254. ACM (2005)
19. Kulkarni, S.S., Wang, L.: MNP: multihop network reprogramming service for sensor networks. In: Proceedings of 25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005), Columbus, USA, pp. 7–16. IEEE (2005)
20. Levis, P., Culler, D.: Mate: a tiny virtual machine for sensor networks. SIGARCH Comput. Archit. News **30**(5), 85–95 (2002). doi:10.1145/635506.605407. http://doi.acm.org/10.1145/635506.605407
21. Levis, P., Culler, D.: The firecracker protocol. In: Proceedings of the 11th Workshop on ACM SIGOPS European Workshop, Leuven, Belgium, p. 3. ACM (2004)
22. Levis, P.A., Patel, N., Culler, D., Shenker, S.: Trickle: a self regulating algorithm for code propagation and maintenance in wireless sensor networks. Computer Science Division, University of California, Berkeley (2003)
23. Lientz, B.P., Swanson, E.B.: Problems in application software maintenance. Commun. ACM **24**(11), 763–769 (1981). doi:10.1145/358790.358796. http://doi.acm.org/10.1145/358790.358796
24. Liu, W., Luo, R., Yang, H.: Cryptography overhead evaluation and analysis for wireless sensor networks. In: Proceeding of Communications and Mobile Computing (CMC '09). WRI International Conference, Kunming, USA, vol. 3 (2009)
25. Panta, R.K., Bagchi, S., Midkiff, S.P.: Zephyr: efficient incremental reprogramming of sensor nodes using function call indirections and difference computation. In: Proceedings of the 2009 Conference on USENIX Annual Technical Conference (USENIX'09), San Diego, USA, pp. 32–32. USENIX Association, Berkeley (2009)
26. Perez, O., Berviller, Y., Tanougast, C., Weber, S.: Comparison of various strategies of implementation of the algorithm of encryption AES on FPGA. In: 2006 IEEE International Symposium on Industrial Electronics, Montreal, Canada, vol. 4, pp. 3276–3280 (2006). doi:10.1109/ISIE.2006.296142

27. Peterson, W., Brown, D.: Cyclic codes for error detection. Proc IRE **49**(1), 228–235 (1961). doi:10.1109/JRPROC.1961.287814
28. Prasithsangaree, P., Krishnamurthy, P.: Analysis of energy consumption of RC4 and AES algorithms in wireless lans. In: International Conference on Anti-Counterfeiting Security and Identification in Communication, San Francisco, USA, pp. 1–5 (2003)
29. Przygienda, T.: Reserved type, length and value (TLV) codepoints in intermediate system to intermediate system. Network Working Group (2002)
30. Reijers, N., Langendoen, K.: Efficient code distribution in wireless sensor networks. In: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA '03), San Diego, USA, pp. 60–67. ACM, New York (2003). doi:10.1145/941350.941359. http://doi.acm.org/10.1145/941350.941359
31. Rifa-Pous, H., Herrera-Joancomarta, J.: Computational and energy costs of cryptographic algorithms on handheld devices. Future Internet **3**(1), 31–48 (2011). doi:10.3390/fi3010031. http://www.mdpi.com/1999-5903/3/1/31
32. Silica: Silica Xynergy-M4 board (2012). http://www.silica.com/product/silica-xynergy-m4-board.html
33. SILICA – The Engineers of Distribution: http://www.silica.com/product/st-stm32-f1f2l1.html (2010)
34. Stathopoulos, T., Heidemann, J., Estrin, D.: A remote code update mechanism for wireless sensor networks. Technical report, DTIC Document (2003)
35. Stathopoulos, T., Mchenry, T., Heidemann, J., Estrin, D.: A remote code update mechanism for wireless sensor networks. Technical report (2003)
36. Stigge, M., Plötz, H., Müller, W., Redlich, J.P.: Reversing CRC – theory and practice (2006). http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05_.pdf
37. STMicroelectronics: RM0090 Reference manual (2014). http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf
38. Toldinas, J., Stuikys, V., Damasevicius, R., Ziberkas, G., Banionis, M.: Energy efficiency comparison with cipher strength of AES and Rijndael cryptographic algorithms in mobile devices. Elektron. Elektrotech. (2011). doi:http://dx.doi.org/10.5755/j01.eee.108.2.134
39. Unterschutz, S., Turau, V.: Fail-safe over-the-air programming and error recovery in wireless networks. In: 2012 Proceedings of the Tenth Workshop on Intelligent Solutions in Embedded Systems (WISES), Klagenfurt, Austria, pp. 27–32. IEEE (2012)
40. Wang, Y., Attebury, G., Ramamurthy, B.: A survey of security issues in wireless sensor networks. IEEE Commun. Surv. Tutor. **8**, 2–23 (2007)
41. Wollinger, T., Guajardo, J., Paar, C.: Security on FPGAs: state-of-the-art implementations and attacks. ACM Trans. Embed. Comput. Syst. **3**(3), 534–574 (2004). doi:10.1145/1015047.1015052. http://doi.acm.org/10.1145/1015047.1015052
42. Xilinx: Efficient shift registers, LFSR counters, and long pseudo-random sequence generators. Application note XAPP 052 Version 1.1 (1996). http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf
43. XNP: http://www.tinyos.net/tinyos-1.x/doc/Xnp.pdf (2003)
44. Yenuguvanilanka, J., Elkeelany, O.: Performance evaluation of hardware models of advanced encryption standard (AES) algorithm. In: Southeastcon, 2008, Huntsville. IEEE, USA, pp. 222–225 (2008). doi:10.1109/SECON.2008.4494289
45. Zhang, J.H., Zhang, N.: A chaos-based key predistribution and management scheme in wireless sensor network. In: International Conference on E-Business and Information System Security (EBISS'09), Wuhan, China, pp. 1–5 (2009). doi:10.1109/EBISS.2009.5137864
46. Zhang, T., Ding, Q.: Design and implementation of CRC based on FPGA. In: 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications (IBICA), Shenzhen, pp. 160–162 (2011). doi:10.1109/IBICA.2011.44
47. Zhao, B., Zhang, H., Li, Z.: A trusted start-up based on embedded system. In: International Conference on Computer and Information Technology, Xiamen, China, vol. 2, pp. 242–246 (2009). doi:http://doi.ieeecomputersociety.org/10.1109/CIT.2009.98

Part III
Building Blocks

Chapter 6
Physically Unclonable Function: Principle, Design and Characterization of the Loop PUF

Zouha Cherif, Jean-Luc Danger, Florent Lozac'h, and Philippe Nguyen

Abstract This chapter presents a novel Physically Unclonable Function PUF called "Loop PUF" (LPUF) which has been studied specifically to be easy to design, lightweight and reliable. The LPUF principle is based on a loop of N controllable delay lines forming a unique ring oscillator. It offers a huge set of challenges as the identity extraction is performed by N measurements, whereas the extraction is made differentially for the other delay-based PUF. This property allows the designer to forge stronger challenge-response protocols or to generate more reliable internal keys. The LPUF concept has been designed and evaluated in ASIC 65 nm technology and in FPGA Virtex-5. The evaluation results show good properties of randomness and uniqueness. As the LPUF output is in integer format and can use numerous challenges, it provides a good base to enhance the steadiness and build reliable authentication protocols. This chapter presents an example of authentication primitive which is very steady in a large range of environmental conditions.

6.1 PUF Background

Physically Unclonable Functions (PUFs) can be defined as a function which returns a characteristic value, or the fingerprint of an integrated circuit. Challenge-Response Pair (CRP) protocol for authentication or cryptographic key generation are two main purposes of PUF. PUF avoids the use of digital memory to store a signature or a key, hence they are well suited in low-cost devices as the RFIDs or smartcards.

Z. Cherif • F. Lozac'h
Telecom ParisTech, 46 rue Barrault, 75013 Paris, France
e-mail: zouha.cherif@telecom-paristech.fr; florent.lozach@telecom-paristech.fr

J.-L. Danger (✉)
Telecom ParisTech, 46 rue Barrault, 75013 Paris, France

Secure-IC, 37 rue Dareau, 75014 Paris, France
e-mail: jeanluc.danger@telecom-paristech.fr; jean-luc.danger@secure-ic.com

P. Nguyen
Secure-IC, 80 avenue des buttes de Coesmes, 35700 Rennes, France
e-mail: philippe.nguyen@secure-ic.com

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_6

The silicon PUF outputs a "response" (or ID) that depends on a control word, called the "challenge". Due to the dispersion of the manufacturing process, the response for a given challenge differs from one PUF to another. There are two main classes of silicon PUFs: the PUFs based on delay comparisons, composed of identical elements, and the PUFs exploiting the initial state of memory blocks.

The first silicon PUF introduced by Gassend et al. [2] is the arbiter PUF. It is a delay based PUF where the delays between two identical controlled paths are compared. From the arbiter PUF derives the XOR PUF, as suggested by Suh et al. [9], and the lightweight secure PUF, as proposed by Majzoobi et al. [7]. These PUFs would allow to mitigate the problem of the modeling attack [8]. The ring-oscillator (RO) PUFs proposed by Suh et al. [9] are based on the comparison between the oscillation frequency of selected pairs of ring-oscillators.

Memory based PUF has been introduced first by Guajardo et al. [3] as SRAM PUFs. Its response is directly related with the state of the SRAM at power up. The disadvantage of these SRAM PUFs is that not all FPGAs supports uninitialized SRAM memory. Therefore, Kumar et al. [6] propose the butterfly PUF that can be used on all types of FPGAs. It works as the SRAM PUF with a memory point based on two flip-flops.

The loop PUF (LPUF) introduced by Cherif et al. [5] and presented here in a in a detailed manner, is a delay based PUF which uses a single ring oscillator to generate the PUF response. This PUF has been characterized in both FPGA and ASIC 65 nm technology and compared with classical arbiter PUF into 18 ASICs platforms. To perform an efficient characterization of PUFs, at least three metrics are necessary: randomness, uniqueness and steadiness.

- **The randomness** gives an estimate of the imbalance between the number of IDs at '0' and the IDs at '1' for all the challenges.
- **The uniqueness** indicates the entropy between two PUFs, either in the same device (intra-uniqueness) or between devices(inter-uniqueness).
- **The steadiness** (or **reliability**) expresses the level of PUF reliability which is reduced by the noise coming from the measurement environment.

Another important metric is **the security** against attacks. The modeling attack [8] is the most powerful against delay-based PUF. It is a mathematical attack which uses machine learning algorithms to reconstruct the elementary delays of delay based PUFs. It is easy to thwart by using lightweight cryptography, as PRESENT, on the challenges or responses of the PUF.

This chapter does not address the security aspect. It mainly focuses on the characterization of other metrics and presents a reliable primitive for authentication.

6.2 LPUF Principle

The loop PUF relies on N identical delay chains ($N \geq 2$) serially connected, each delay chain being composed of M controlled delay elements ($M \geq 1$). Each delay element is merely a multiplexer generating a delay specific to the control bit.

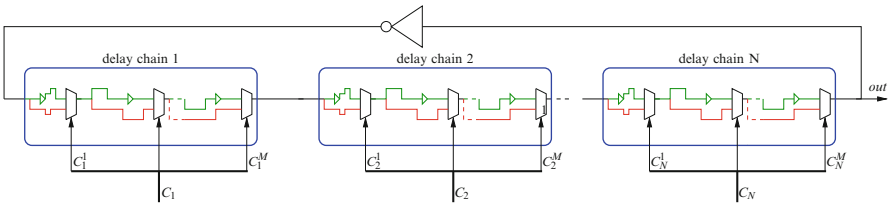


Fig. 6.1 Loop PUF structure

Table 6.1 Number of challenges

		M									
		2	3	4	5	6	7	8	10	12	16
$Ch_{arbiter}$		4	8	16	32	64	128	256	1,024	4,096	65,536
Ch_{LPUF}	$N = 2$	4	13	40	121	364	1,093	3,280	29,524	∼250 K	∼21 M
	$N = 3$	4	44	360	2,680	19,244	∼130 K	∼1 M	∼45 M	∼2 G	∼5,000 G

The architecture is illustrated in Fig. 6.1. The routing internal to the delay chain can be random, without routing constraints. This is particularly appreciated in FPGA where it is hardly possible to constrain the two paths of Look Up Table (LUT) inputs. The only constraint is to copy/paste the delay chain which is quite easy in any technology. For instance in FPGA the routing between two Configurable Logic Blocks (CLB) of Xilinx, or Logic Array Blocks (LAB) of ALTERA can be similar. When closed by an inverter, this structure forms a single ring oscillator. The oscillation frequency depends on the "challenge" which is the concatenation of the N control words $C_1, \ldots, C_N$ applied to the N delay chains. For a given challenge ch_i, the number of oscillations T_i are counted during a fixed measurement window. As the delay chain are structurally identical, the permutation of the N control words internal to the challenge should not change the oscillation frequency. The challenges corresponding to all the permutations are "equivalent challenges" as they should give the same frequency. By definition the challenges are "equivalent" if they are elements of a set of M words of N bits (for N lines of M elements), each word having the same Hamming Weight.

Because of the CMOS variability in physical devices, the measured T_i are slightly different between the equivalent challenges, thus creating a signature of the device. The PUF response can be the time difference $\Delta_T = T_1 - T_2$ measured with two equivalent challenges $ch1$ and $ch2$. It can also be the sorting of measured times for all the equivalent challenges. It has been shown in [5] that the LPUF allows the user to generate a huge set of challenges. However a subset corresponding to those having the biggest Hamming distance should be kept in order to mitigate correlation effects. Table 6.1 shows the maximum number of possible challenges for $N = 2$, $N = 3$ and for different values of M.

6.3 Evaluation Metrics

Section 6.1 introduced the three main qualities that a PUF has to satisfy: randomness, uniqueness and reliability. There is no standard to evaluate the PUF performance, however, a few evaluation methods have been proposed recently.

In 2010, Hori et al. [4] proposed a method to evaluate all kinds of PUFs. It is based on the statistical processing of the **logical IDs**. To elaborate an accurate characterization, exhaustive tests are needed. This method is therefore time consuming.

Another method presented in 2011 by Cherif et al. [1] is applicable only for delay PUFs. It gives an estimate of the PUF characteristic with the statistical evaluation of **delay elements**. This method is faster because it does not need to run many challenges and it relies only on delay measurements. It is well suited to PUF designs in ASIC as the delay extraction can be done after the layout stage, allowing the designer to evaluate the PUF before fabrication. Hence, it is not adapted to FPGA design unless their structure is modified to permit the delay measurement. We used the Hori method since it is applicable for all PUF types. Below, we describe the three main metrics to evaluate the performance indicators of a PUF as proposed by Hori et al. [4].

- **Randomness metrics**
  It uses the **min-entropy** H of a bit sequence of the PUF response when applying different challenges to evaluate the ability of the PUF to produce as much 0 as 1. When the PUF response is perfectly balanced, its randomness (maximum min-entropy) is equal to 100 %.
- **Steadiness Metrics**
  Considering the **min-entropy** H of a bit sequence obtained when applying the same control word T times, the steadiness metric is considered as being 1-H. When a PUF response is stable, the steadiness of the PUF is the highest (closed to 100 %). We distinguish two types of steadiness evaluation: The steadiness of a PUF under normal environmental condition and its steadiness when running under different operating conditions (temperature variation, power supply voltage variation, etc.).
- **Uniqueness metrics**
  The uniqueness metric is based on the normalized Sum of Hamming Distance (**SHD**) of the possible ID-combinations, when applying the same challenge set to different PUFs (either located in the same device or not). The higher is the SHD (close to 100 %), the greater is the uniqueness of the PUF.

6.4 Design and Evaluation Platform

The platform is composed of a mixed PUF, i.e. arbiter PUF and Loop PUF, called PUFmix. This PUFmix block contains the same basic delay elements in order to guaranty a fair comparison between Arbiter PUF and Loop PUF. It is designed

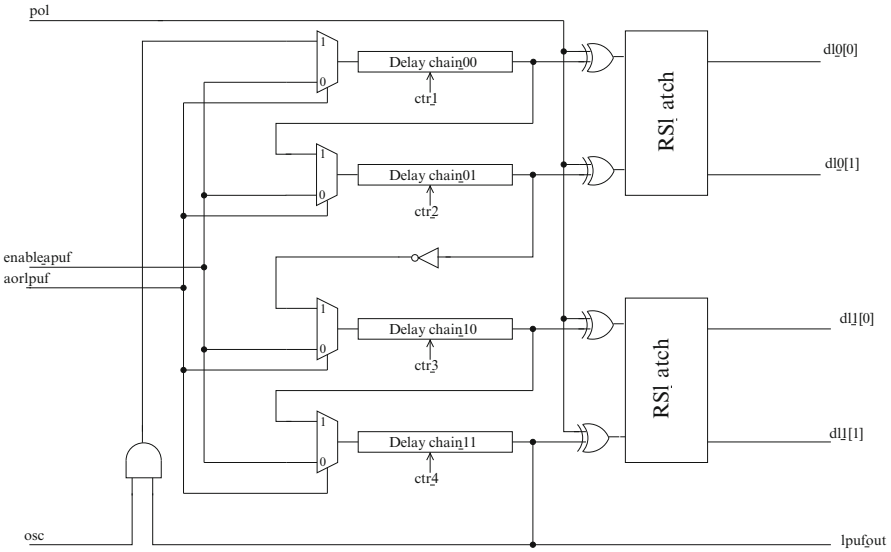


Fig. 6.2 PUFmix design

on two platforms: FPGA and ASIC in 65 nm technology, each platform having 49 PUFmix structures. The intra-device (using 49 PUFs) and inter-device (using 18 devices) characteristics have been studied for both PUFs. The characteristics results use the three metrics presented in Sect. 6.3: randomness, uniqueness and steadiness. Figure 6.2 shows the PUFmix structure designed on both FPGA and ASIC platforms. Four delay chains are used on the PUFmix design to make three independent PUFs:

- Arbiter PUF #1 (uses the two upper delay chains).
- Arbiter PUF #2 (uses the two bottom delay chains).
- Loop PUF (uses the four delay chains).

Each delay chain in composed of M=16 basic delay elements. At the end of each chain, a buffer to equilibrate the end charges of the four chains. A multiplexer is used before each delay to select the operating mode of the design (arbiter or loop PUF) depending on the $aorl\_puf$ signal. Using four delay chains, the loop PUF generates 4 delays which can be sorted by an external controller to generate a 4-bit response. However, each arbiter PUF generates a 1-bit response.

6.4.1 ASIC Platform

The ASIC prototype has been designed in technology STM 65 nm. The layout with the 49 PUFmix organized in 7×7 matrix is shown in Fig. 6.3. The PUFmix implementation uses 215 standard cells (343 gates). The delay lines have been

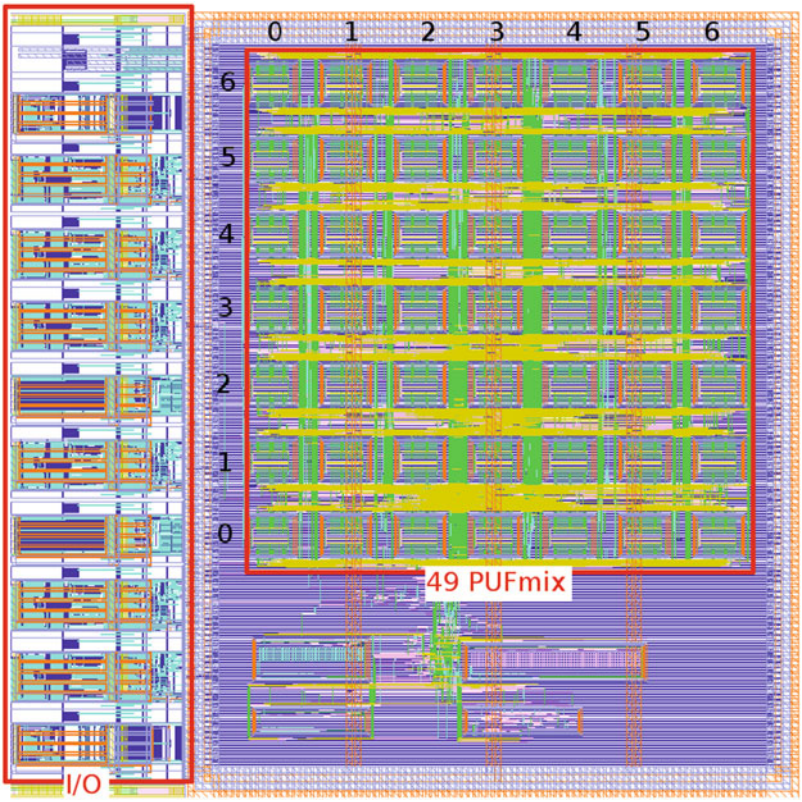


Fig. 6.3 ASIC layout

exactly copied and pasted four times. All output signals are isolated from the I/O pins using buffer in order to keep equilibrate timing performances for the arbiter PUF. Hence, each PUFmix occupies $50.8 \times 44.28\,\mu\text{m} = 2{,}249.424\ \mu\text{m}^2$ in the ASIC. Identical routed delay chains are designed using an automatic script in order to ensure identical paths. For the arbiter PUF, a specific place and route has been done for the $RS\_latches$ in order to obtain a balanced structure.

A test board, shown in Fig. 6.4, has been designed with controllable power supply pins for the ASIC core. This allows the designer to evaluate the impact of voltage variation on the steadiness characteristic. The communication with the device is done via an UART module which has its own clock and Power supply.

6.4.2 FPGA

A 168 PUFmix design has been implemented on a Xilinx Virtex-5 vlx50t FPGA embedded on a Digilent Genesys development board. To obtain such a circuit, specific methodology has been used to achieve the two main constraints:

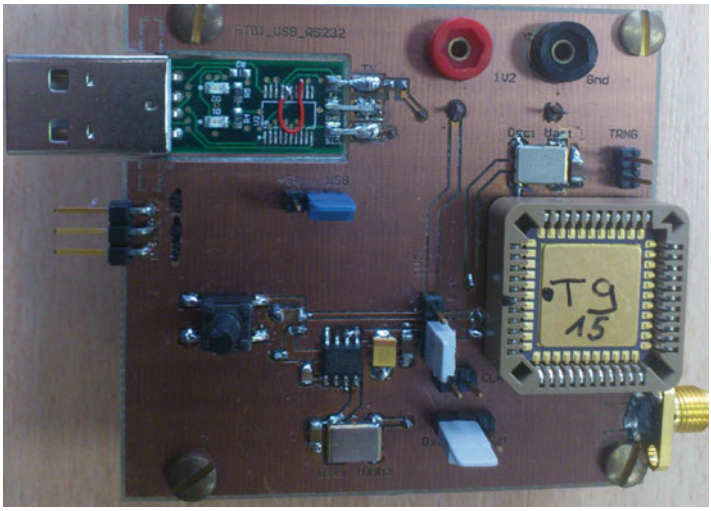


Fig. 6.4 Test board

- The four delay chains have to be exactly similar in terms of resource placement and routing.
- The PUFs have to be cloned.

Achieving these constraints is not possible at RTL level only. It is necessary to apply a methodology with specific Xilinx "hard macro" and constraints:

- *Hard Macro*: It is a placed and routed design part, which does not contain any input/output buffers (IOB). A *hard macro* can be instantiated more than once in a circuit. Each instance is a clone of the original module reproducing its placement and routing. *Hard macros* are stored in NMC files, a Xilinx file format is quite similar to *NCD* files, already used to describe classic netlists. Custom scripts have been developed to automatically generate *hard macros*.
- *Primitive instantiation*: We can include some Xilinx specific primitives in HDL code to infer for example LUTs, and define their truth table and inputs mapping. Those primitives are described in the Virtex-5 Libraries Guide for HDL Designs [10].
- *Xilinx constraints*: Xilinx tools offer the possibility to attach to a particular design some constraints and attributes which affect the implementation like logical, physical or mapping constraints. Undesired place or route optimizations can also be forbidden.

The detailed steps necessary to meet the Place/Route constraints are the following:

1. Primitive instantiation technique is first used to design a delay element chain where each element is a LUT which is configured as a MUX21. Elements are manually placed from left to right, on a same slice row. Manual placement

is done taking advantage of design constraints that Xilinx provide us, like "LOC", "RLOC". Others constraints to prevent Xilinx design flow tools from making undesired optimizations are also added, like "SAVE NET FLAG" or LOCK_PINS". The two first LUT inputs are logically connected to the previous element output, while the 3rd input is reserved to be connected to one control signal (ctr). Virtex-5 slices consist of four LUTs, so one chain of 16 elements occupies four slices. After chain is designed, our custom scripts are computed, and a "chain" Hardmacro is created, preserving its placement and routing.
2. Then, chain Hardmacros are instantiated on successive rows. Additional logic operators used for both arbiter and loop PUF are manually placed. Also, routing is not constrained, so differences may exist between the two arbiter PUFs. Next, a PUFmix Hardmacro is created.
3. Finally, all PUFmix Hardmacro instances are manually placed on FPGA matrix.

Every instance occupies 24 slices, which is about 0.4 % of slice resources. Final Layout is shown in Fig. 6.5. We can notice in Fig. 6.5 that some columns are not usable for PUFmix as it requires four adjacent columns whose slice type is M, L, L, L, respectively. As for some columns the sequence is M, L, M, L,

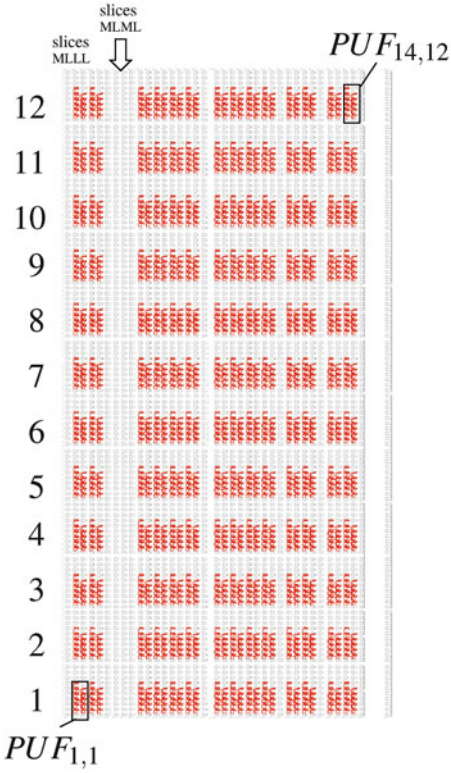


Fig. 6.5 Layout of FPGA design where 168 PUFmix Hardmacros are instantiated

hence the PUFmix cannot be placed here. In order to fairly compare the PUFmix performance when designed in both platforms, only 49 PUFmix are considered on the FPGA.

6.5 Experimental Results

6.5.1 Intra-device Evaluation

In this section, we compare the PUFmix intra-device performance when designed on two different platforms (FPGA and ASIC) using the CMOS 65 nm technology. We randomly select an ASIC device among the 18. Three performance indicators are investigated in order to evaluate the PUFmix (2 arbiter PUFs and 1 loop PUF): randomness, uniqueness and steadiness.

To characterize the randomness of each arbiter and each loop PUF, we compute the min-entropy of the bit response sequence when applying the same set of different control words to the 49 PUF instances. Figure 6.6a, b show the average of intra-device evaluation results of the 49 arbiter PUF #1 and the 49 arbiter PUF #2, respectively on the two used platforms. On the FPGA, the randomness of the arbiter PUF #1 is 0 %. This means that, despite checking the routing time at design process, there is a big bias between the two parallel paths due to imperfect routing. The bit response of the PUF is stable (always at '0' or '1') even when changing the control word. The intra-device evaluation of the arbiter PUF #2 shows that the bias on FPGAs is reduced and the randomness of the arbiter PUF #2 increases to 25 %. However, on ASIC, the two arbiter PUFs present almost the same performance results. Then, we can conclude that, due to manual routing, the arbiter PUF design on ASIC is slightly better in terms of randomness (around 30 %) than on FPGA.

Using the min-entropy of the bit response sequence obtained when introducing the same set of challenge $T = 128$ times, we assess the steadiness of the PUF. Figure 6.6a, b show the average steadiness of the 49 arbiter PUF #1 and the 49 arbiter PUF #2, respectively. Since there is a bias on the design of the arbiter PUF structures (poor randomness), we cannot judge the steadiness which is around

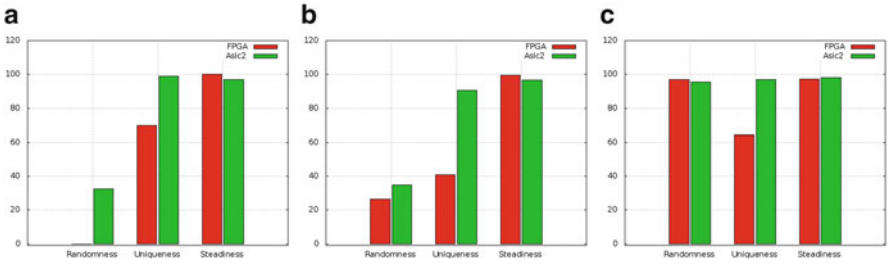


Fig. 6.6 Intra-device evaluation. (**a**) Arbiter PUF #1. (**b**) Arbiter PUF #2. (**c**) Loop PUF

100 % on both platforms. Therefore, the steadiness of the arbiter PUFs cannot be investigated even on the ASIC platform since it is influenced by the low randomness characteristic (around 30 %). Our results show (Fig. 6.6c) that, on both platforms, the loop PUF presents a good randomness characteristic (around 100 %), since there is no need for routing constraints. In this case the steadiness of the PUF can be investigated. And the average steadiness of the 49 loop PUFs is around 98 % on ASIC and FPGA platform. This proves that the loop PUF design is very reliable independently of the used platform.

The intra-device uniqueness of the PUFs response is studied using the SHD metric. Although both platforms are built with the CMOS 65 nm technology, due to the noise of designed and unused components on the FPGA, the extraction process is better on ASIC. This makes the uniqueness of the designed PUFs (arbiter and loop PUFs) better on ASIC than FPGA.

6.5.2 Deeper Analysis on ASICs

In this section, we start our investigation by the evaluation of the randomness and the steadiness of the arbiter and the loop PUF structures when varying operating conditions. Then we study the inter-device characteristics of the PUFs when placed in the same places in different ASICs.

6.5.2.1 Randomness Analysis

We randomly choose a set of 1,024 challenges for the evaluation of the randomness of the arbiter and the loop PUF instances under different operating conditions. As proposed by Hori et al. [4], we compute the min-entropy of the obtained bit sequence to evaluate the randomness of each PUF when applying the 1,024 fixed challenges.

Figure 6.7 presents the mean value of the randomness percentage obtained for the 49 PUFs for each structure when performing under different temperatures. The results show that both arbiter PUFs present poor randomness characteristics. The highest randomness value for the arbiter PUFs is 34.03 %. It is obtained when performing under normal operating conditions for the arbiter PUF #2. However, the best randomness value for the loop PUF is obtained under the nominal operating conditions (98.92 %).

When varying the supply voltage ($\pm$10 % of the nominal 1.2 V), the evaluation of the randomness of two arbiter PUFs and the loop PUF shows that the randomness of both structures is independent from the supply voltage variation (Fig. 6.7). The loop PUF is far better than the arbiter PUF in terms of randomness even when varying the supply voltage. The higher randomness value for the arbiter PUFs is 34.73 %, however, the lower value for the loop PUF is 98.94 %.

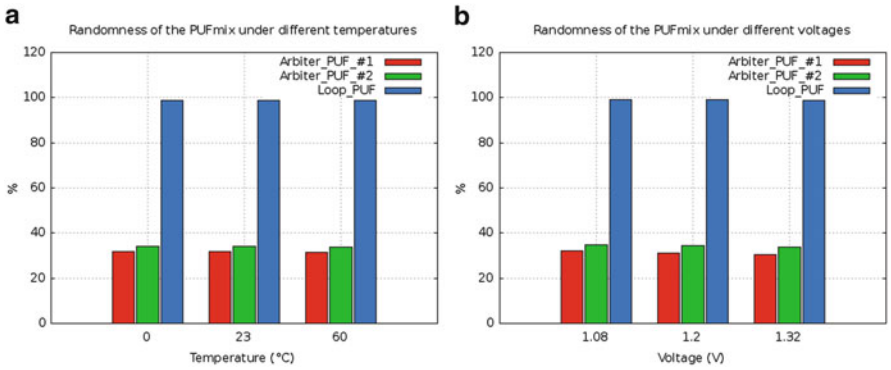


Fig. 6.7 PUFmix randomness evaluation. (**a**) Temperature impact. (**b**) Voltage impact

We conclude that there is a big gap of the randomness performance between the arbiter and the loop PUF structures on the ASIC platform. The loop PUF is better than the arbiter PUFs when varying the temperature (0, 23 and 60 °C) and the supply voltage (±10 % of the nominal 1.2 V).

As discussed in Sect. 6.5.1, the poor randomness of the arbiter PUFs can be explained by the imperfect balance between the two paths. The imperfect balance of paths influence negatively the randomness of the arbiter PUF and positively on the steadiness of the PUF. Indeed a very poor randomness increases the steadiness as the PUF output has more probability to be steady if it as often the same value. Therefore, the evaluation of the steadiness is not appropriate in the case of low values of the randomness (for arbiter PUFs in our case).

6.5.2.2 Steadiness Analysis

The steadiness, or reliability, is the property that a PUF always generates the same response even when varying operating conditions, supply voltage and temperature. By using the Hori [4] metrics, we determine the average steadiness of the response of the 49 PUFs designed on a randomly selected ASIC when applying a fixed random challenge. This analysis is done under different ambient temperatures (0, 23 and 60 °C) and supply voltages (±10 % of the nominal 1.2 V). As mentioned above, the reliability of the arbiter PUFs is not very relevant since it is very influenced by the bias between the paths (low randomness).

When varying the temperature, the loop PUF steadiness remains stable. This means that loop PUF instances does not depend on the operating temperature. However, the supply voltage variation influences the loop PUF steadiness performance. At nominal supply voltage the loop PUF steadiness is the highest (= 98.26 %). when varying the operating voltage (±10 % of the nominal 1.2 V), the steadiness of the loop PUF is slightly reduced (around 92.5 %). The results illustrated in Fig. 6.8a, b show that the loop PUF instances have a good steadiness (higher than 92 %), which

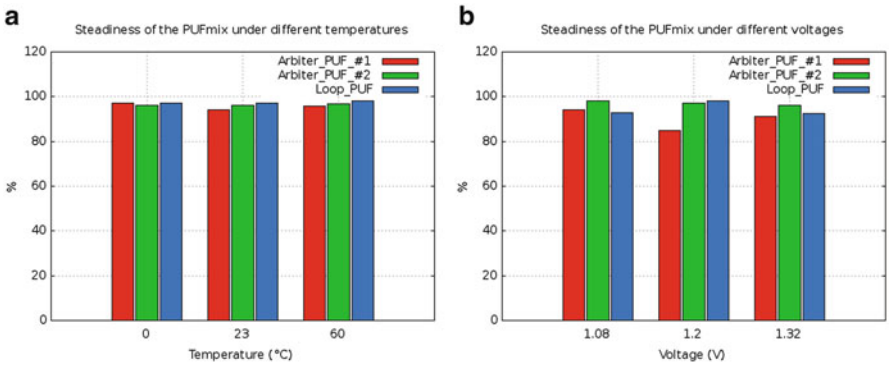


Fig. 6.8 Loop PUF steadiness evaluation. (**a**) Temperature impact. (**b**) Voltage impact

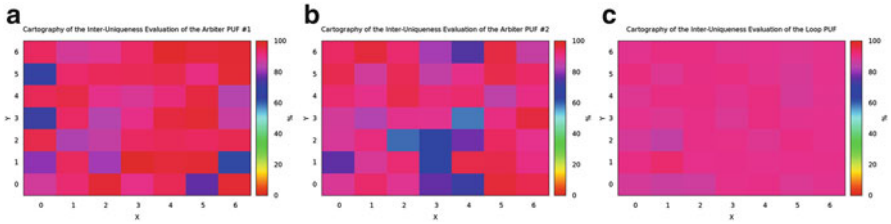


Fig. 6.9 Inter-uniqueness evaluation. (**a**) Arbiter PUF #1. (**b**) Arbiter PUF #2. (**c**) loop PUF

can be handled using a lightweight error correction schemes. We conclude that the loop PUF are arbiter PUF instances have a good level of steadiness against variation of temperature and supply voltage, but the arbiter PUF takes advantage of its low randomness.

6.5.2.3 Inter-uniqueness Evaluation

The inter-uniqueness shows the ability of a PUF to produce different responses when designed in the same place in identical devices when applying the same challenge under normal operating condition. We use the Hori method, which is based on the computation of the SHD of equivalent bit response.

Figure 6.9a–c present a cartography of the inter-uniqueness evaluation over the 18 ASICs of the arbiter PUF #1, the arbiter PUF #2 and the loop PUF, respectively.

Our results show that the loop PUF has the best average inter-uniqueness characteristics. Almost all the 49 loop PUFs have an inter-uniqueness around 90 %. However, the variance of the inter-uniqueness of the 49 arbiter PUFs #1 and the 49 arbiter PUFs #2 is very large. The PUF located in cell (1,0) has the least inter-uniqueness value of 86.08 %. However, both arbiter PUFs present lower inter-uniqueness characteristics than the loop PUF. The lowest values are 61.83 and 57.90 % for the arbiter PUF #1 (located in cell (6,1)) and the arbiter PUF #2 (located in cell (4,3)), respectively.

We conclude that the inter-uniqueness of the loop PUF is better than the inter-uniqueness of both arbiter PUFs. Regardless of its location on the ASIC, the loop PUF structure presents similar inter-uniqueness performance.

6.6 Loop PUF for Authentication

Due to the combination of their properties of uniqueness and steadiness, PUFs are presented as an innovative primitive to derive secret from complex physical characteristics of ICs. They are proposed to be used for low cost authentication of ICs and the generation of cryptographic keys. For instance Suh et al. [9] propose methods based on "Challenge-Response Protocol" (CRP) to use the ring-oscillator PUFs for low cost authentication and cryptographic key generation. They challenge-response table is first recorded by the authentication server. Then at each authentication operation, the trusted server has to select only one recorded and not previously used challenge. Finally, to check the authenticity of the IC, the response of the PUFs for the same selected challenge have to meet **exactly** the response saved on the recorded table. Using this method, we consider that the PUF response is perfect and no errors can occur, which is not true since the error probability of the proposed PUF is not null. The Loop PUF outputs an integer value which represents the delay or the frequency of the loop during a fixed period. From this output, we can either generate a binary response (as for the others PUFs on the literature) or use directly the integer output which is much more precise and significant. In this chapter, we propose a method that takes advantage from this integer output for device authentication purpose. It is based on the measurement of physical values of delay elements. These physical values are used to authenticate devices since they are much more precise than the binary response of the loop PUF. The proposed method can be divided into two principle steps:

1. Enrollment step.
2. Authentication step.

6.6.1 Enrollment Step

The main rationale of this step is to generate the reference vectors which are used as a helper data, at the authentication step, to analyse the IC authenticity. It consists in measuring the delays of basic loop PUF delay elements. At the end of this stage, the number of M reference vectors are saved, with M the number of basic delay elements included on a loop PUF delay chain. Therefore, we propose to:

1. Measure the global delay d_0 of the loop PUF when all delay elements are set to '0' ($C_i^j = 0$ with $i \in [1, N]$ and $j \in [1, M]$).
2. Measure separately the delay $d_{i,j}$ of each delay element j on each delay chain i when $C_i^j = 1$.

3. Construct the measurement vectors Ref^j with $j \in [1, M]$ such that: $\mathsf{Ref^j} = [(\mathsf{d_{1,j}} - \mathsf{d_0}); (\mathsf{d_{2,j}} - \mathsf{d_0}); (\mathsf{d_{3,j}} - \mathsf{d_0}); (\mathsf{d_{i,j}} - \mathsf{d_0}); \cdots ; (\mathsf{d_{N,j}} - \mathsf{d_0})]$.

This enrollment procedure is performed only once. The reference vectors which are generated are further compared to the generated vectors during the authentication step.

6.6.2 Authentication

It consists in checking if the generated measurement vectors are correlated or not with the references vectors. To do so, we propose to use the Pearson correlation coefficient. The proposed method is based on four essential steps. The three first steps are identical to those performed during the learning stage.

1. Measurement of the global delay d_0 of the loop PUF when all delay elements are set to '0' ($C_i^j = 0$ with $i \in [1, N]$ and $j \in [1, M]$).
2. Measurement of the delay $d_{i,j}$ of each delay elements j on each delay chain i when $C_i^j = 1$.
3. Construction of the vectors X^j with $j \in [1, M]$. $\mathsf{X^j} = [(\mathsf{d_{1,j}} - \mathsf{d_0}); (\mathsf{d_{2,j}} - \mathsf{d_0}); (\mathsf{d_{3,j}} - \mathsf{d_0}); (\mathsf{d_{i,j}} - \mathsf{d_0}); \cdots ; (\mathsf{d_{N,j}} - \mathsf{d_0})]$.
4. Computation of the Pearson correlation coefficient (see Eq. 6.1) between each normalized pair of reference and measurement vector $corr^j_{(Ref^j, X^j)}$ with Ref^j the reference vector already recorded.

The Pearson coefficient used as a metric to authenticate a device is given by:

$$\mathsf{corr_{(Ref,X)}} = \frac{1}{\mathsf{M}} \sum_{\mathsf{j=1}}^{\mathsf{M}} \mathsf{corr^j_{(Ref,X)}} = \frac{1}{\mathsf{M}} \sum_{\mathsf{j=1}}^{\mathsf{M}} \frac{\sum_{\mathsf{i=1}}^{\mathsf{N}} (\mathsf{ref_i^j} - \hat{\mathsf{ref}}^{\mathsf{j}})(\mathsf{x_i^j} - \hat{\mathsf{x}}^{\mathsf{j}})}{\sigma_{\mathsf{ref}} - \sigma_{\mathsf{x}}}. \tag{6.1}$$

Using this metric for device authentication, we take into account both offset and scaling phenomenon that can affect the PUF response (e.g. by temperature variations). The correlation coefficient is equal to '1' if one of the variables is an increasing function of the other variable, to '−1' in when the function is decreasing. Intermediate values provide information on the degree of linear dependency between two variables. The correlation between variables is strong when the correlation coefficient value is closer to the extreme values '−1' and '1. A correlation coefficient of 0 means that the variables are not correlated.

6.6.3 Experimental Results

The tests have been carried out on 16 ASICs, each one embedding 49 loop PUFs. Hereafter we present first the authentication results when identifying a PUF among others embedded in the same IC (authentication intra-ASICs) either in

ambient temperature or when varying it. Then we study the performance of the authentication metrics to identify a PUF among others situated in the same location in identical ICs (authentication inter-ASICs) at nominal environmental conditions. The experimental setup is composed of four delay chains ($N = 4$). Each one contains 16 delay elements ($M = 16$). As the routing of the 16 elements is identical, we can consider a unique delay chain composed of 64 elements ($N = 64$ and $M = 1$). The steps (1)–(3) presented above are performed $T = 128$ times in order to assess the reliability of the PUF response. The first test of the measurement vectors X^j is a reference vector referred to Ref^j. And then the mean correlation coefficient is computed as described on the fourth and last step of our authentication step.

6.6.3.1 Intra-device Authentication

In order to verify the ability of the loop PUF to be used for authentication purposes using the proposed method, we study the intra-ASIC correlation on the 16 ASICs.

Figure 6.10 shows the mean value of the correlation coefficients between all PUFs situated in the same device. We note that when we compare a PUF response with its responses (different tests), we have a high correlation (closed to 1). However, when the PUF responses are compared to those of another PUF existing in the same die, the worst case correlation coefficient does not exceed in its absolute value the 0.5 which is very low.

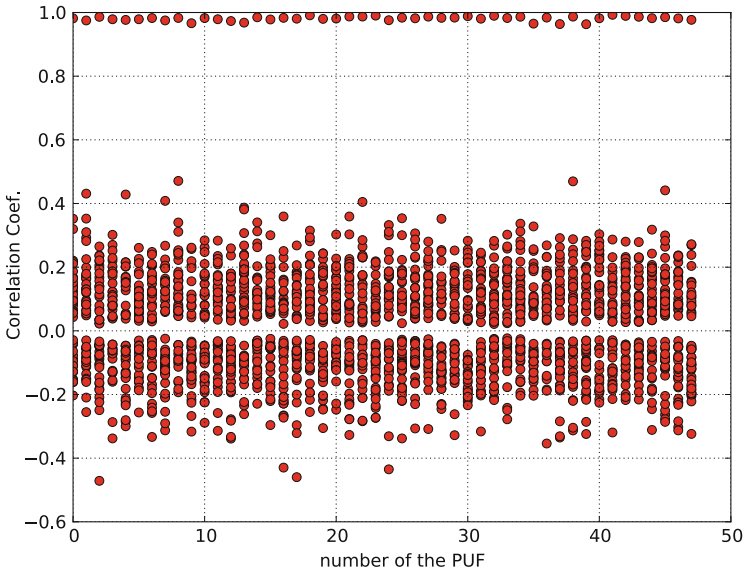


Fig. 6.10 Intra-ASIC mean correlation results

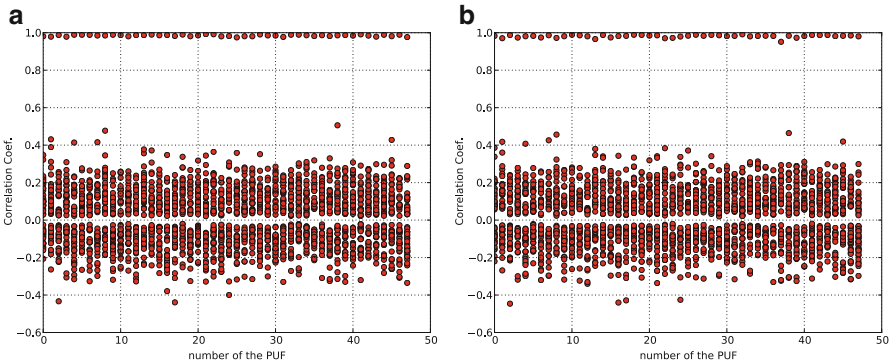


Fig. 6.11 Intra-ASIC mean correlation results at different temperatures. (**a**) Temperature = 0 °C. (**b**) Temperature = 60 °C

Even when decreasing the temperature to 0 °C or increasing it to 60 °C, using our method (the Pearson correlation coefficient), the loop PUF presents good performance to be used for device authentication purposes. The correlation coefficient at different environmental conditions does not exceed in its absolute value the 0.5. This proves that our metric take into account the scaling phenomenon caused by the temperature variation. Figure 6.11a, b illustrate the intra-device authentication results when varying the temperature. We note that the results obtained at 60 °C are the best. This can be explained by the fact that the reliability of the loop PUF increases with the temperature due to the dilation of the delays of delay elements.

6.6.3.2 Inter-device Authentication

To evaluate the ability of the loop PUF to be used for device authentication using the correlation method. We propose to evaluate the correlation degree between each PUF (through its reference vector) and its equivalent (PUFs having the same place) in the other ASICs. Figure 6.12 shows that we are able to distinguish a PUF from another one even when placed in the same place on different ASICs. In the worst case, the correlation coefficient does not exceed in its absolute value the 0.5 which let the error interval very large.

Due to the accuracy of the loop PUF output and the method that we propose, we are able to distinguish a PUF form other similar PUFs either implemented in the same ASIC or placed at the same place in different ASICs without post processing schemes and with a large error margin.

The authentication procedure is not time consuming. The authentication time is linearly dependent with the number of basic delay element of the loop PUF. In our case, since the loop PUF is composed of 64 basic delay elements, we need

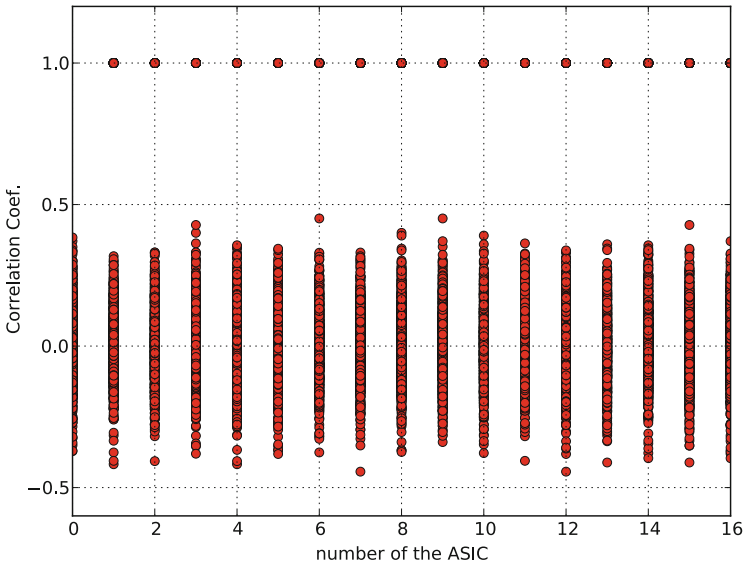


Fig. 6.12 Inter-ASICs correlation results

65 measurements to authenticate a PUF. One measurement when all the control bits are set to '0', and 64 others when activating a single delay element per measurement. With a clock system frequency of 100 MHz, we are able to perform all needed measurements within $1.08\,\text{ms} = 65 * T_{puf}$ which is very low. The metric computation (the fourth step of our method) needs 0 ms to be done. Then the overall authentication procedure of the PUF takes about 1.08 ms for a loop PUF of 64 delay elements.

6.6.3.3 Discussion About the Robustness Against Attacks

In order to secure the transmission of the PUF response (IC identifier) from man in the middle attacks, replay attacks and modeling attacks, a cryptographic layer should be added to the PUF system. The traditional solution to thwart these attacks is to provide a secure challenge-response authentication protocol while exchanging the IC identifier. However this extra logic should not be too complex to harm the low-cost interest of the PUF. Figure 6.13a, b show an example of countermeasure against replay attacks. The cryptographic layer takes advantage of a **Hash** function and a **cryptographic nonce** authentication protocol. This protocol should be both ways if the server is not trusted.

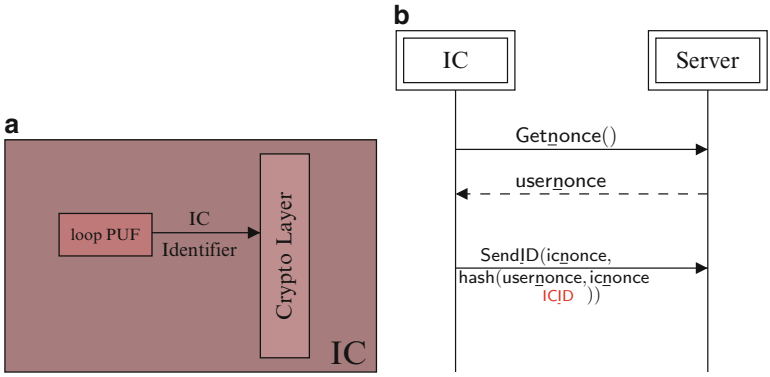


Fig. 6.13 Principle of the secured PUF authentication system. (**a**) PUF authentication system. (**b**) Authentication protocol

Conclusions

The Loop PUF has been introduced and characterized. It has been shown that this structure is easy to design and offers a huge number of challenges as it is based on sequential comparisons on N delay chains. Consequently this permits low-cost methods for authentication and key extraction. The performances have been evaluated and compared with arbiter PUF by means of FPGA and ASIC platforms in CMOS 65 nm technology. The loop PUF presents better performance than the two arbiter PUFs even when varying operating conditions. It has a stable random response, and when placed on different places of the ASIC, it presents similar uniqueness performance. Hence, the LPUF also offers a good base for reliable authentication protocols and key generation. Its output in integer format allows the designer to take advantage of signal processing operators, as the correlation, to generate a unique and steady response whatever the conditions.

References

1. Cherif, Z., Danger, J.L., Bossuet, L.: Performance evaluation of physically unclonable function by delay statistics. In: NEWCAS, Bordeaux (2011)
2. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: Silicon physical random functions. In: ACM Conference on Computer and Communications Security, Washington, DC, USA, pp. 148–160 (2002)
3. Guajardo, J., Kumar, S.S., Schrijen, G.J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: CHES, Vienna. Lecture Notes in Computer Science, pp. 63–80. Springer (2007)

4. Hori, Y., Yoshida, T., Katashita, T., Satoh, A.: Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs. In: International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, pp. 298–303 (2010). doi:http://doi.ieeecomputersociety.org/10.1109/ReConFig.2010.24
5. Jouini, Z.C., Danger, J.L., Guilley, S., Bossuet, L.: An easy to design PUF based on a single oscillator: the loop PUF. In: DSD'12, Izmir, Turkey (2012)
6. Kumar, S.S., Guajardo, J., Maes, R., Schrijen, G.J., Tuyls, P.: The butterfly PUF: protecting IP on every FPGA. In: Tehranipoor, M., Plusquellic, J. (eds.) HOST, pp. 67–70. IEEE Computer Society (2008)
7. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Lightweight secure PUFs. In: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, ICCAD'08, San Jose, USA, pp. 670–673. IEEE, Piscataway (2008). http://portal.acm.org/citation.cfm?id=1509456.1509603
8. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS'10, Chicago, USA, pp. 237–249. ACM, New York (2010). doi:http://doi.acm.org/10.1145/1866307.1866335
9. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: DAC, San Diego, USA, pp. 9–14 (2007)
10. Xilinx: Virtex-5 libraries guide for hdl designs. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/virtex5_hdl.pdf

Chapter 7
Physically Unclonable Function: Design of a Silicon Arbiter-PUF on CMOS 65 nm

Guillaume Reymond and Jacques J.A. Fournier

Abstract This chapter addresses some of the practical difficulties when designing a silicon-PUF based on CMOS technology. When designing such a PUF, and particularly in the case of an arbiter-PUF, particular care should be taken during the place and route phase. To ensure the properties of the PUF, full-custom back-end design may be required. This chapter give details on the design phase considering the example of an arbiter-PUF made of several switch boxes. Each of them can be configured by the challenge, and an arbiter (NAND2 – RS-latch) to decide of the output of the PUF. The technology node targeted for the ASIC implementation is the CMOS 65 nm.

7.1 Arbiter-PUF Background

The successive chemical and physical operations involved in the manufacturing of CMOS circuits create slight variations between instances of identical transistors. This variability allows designing intrinsic silicon PUF. Among the effect of process dispersion, the arbiter-PUFs exploit the difference between propagation delays through similar datapaths. Because silicon PUF can be implemented with a standard manufacturing process, they can be integrated with the CMOS logic.

The first silicon delay PUF was introduced by Gassend et al. [1]. It is based on an arbiter element which compares the delays between two paths composed of M controlled delay elements. An arbiter is used to determine which path is the shortest, by determining on which path the rising edge arrives first. The whole architecture is illustrated in Fig. 7.1.

From a design point of view, the arbiter-PUF requires a specific attention to balance the delays between the two paths. Indeed each pair of paths – the two straight ones and the two crossed ones – needs to be perfectly balanced to prevent from introducing a permanent bias.

G. Reymond (✉) • J.J.A. Fournier
CEA, CTReg, DPACA/LSAS, F-13541 Gardanne, France
e-mail: guillaume.reymond@cea.fr; jacques.fournier@cea.fr

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_7

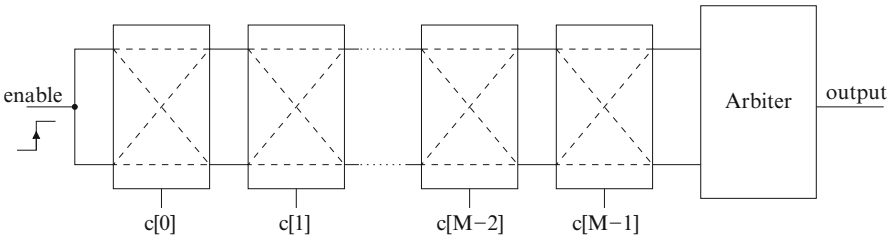


Fig. 7.1 Arbiter-PUF

7.2 Targeting the CMOS 65 nm Technology

The arbiter PUF is implemented using the CMOS 65 nm technology, as a full-custom design: every transistor is placed and routed manually to ensure the symmetry of the block. The PUF is then designed as a standalone IP, and this block is then integrated into a standard design flow for chip finishing: design of the IO ring, place and route of the IPs. We choose not to use standard cells for designing of the following reasons:

- Standard cells paths from inputs to outputs are optimized in terms of performances, but the layout is not balanced between two equivalent datapaths. This is the case for MUX 2-1 and RS-latch. These unbalanced paths may result in a bias in the final design.
- Routing butted standard cells may also result in asymmetry in metal layers.

To facilitate the final integration step, the cells are designed according to the framework of the standard cells. Thus several tasks are made easier:

- Addition of filler cells with N-Well and P-Well straps
- Integration of standard cells fillers – for density consideration
- Meshing of GND and VDD

The framework of a standard cells design is shown Fig. 7.2 and has the following characteristics:

- Total height: 3 μm
- GND and VDD rails height: 0.56 μm

7.3 Design of a Switch Box

The switch box is the configurable element of the arbiter-PUF. Because of the process dispersion, the two configurations result in unpredictable distinct propagation delays for inputs *in1* and *in2*. The functionality of a switch box may be described by two multiplexers controlled by the challenge *c*, as shown Fig. 7.3.

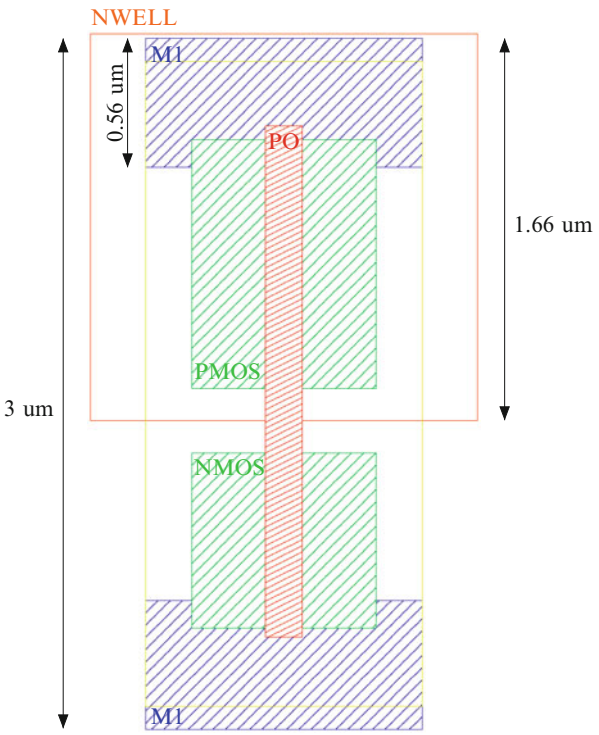


Fig. 7.2 Standard cell framework for CMOS 65 nm

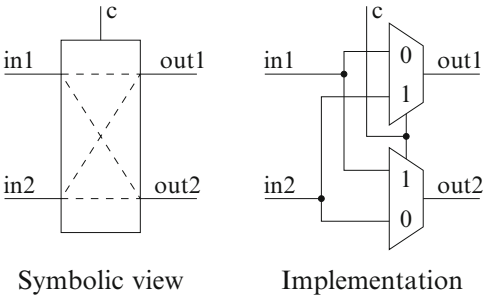


Fig. 7.3 Symbolic view and implementation of a switch box

Based on the design of pass gates used in the MUX21 standard cell, the switch box can be implemented according to the schematic presented Fig. 7.4. Table 7.1 gathered the parameters for the transistors used in the switch box.

The transistors P0 and N0 of the inverter are sized with two fingers for reasons of symmetry. A particular care is required during the place and route phase, in order to keep the design symmetrical. In particular, the two paths have to be balanced regardless of the configuration input c. When $c = 0$, the input *in1* goes through the pass gate P1 – N1 to exit by *out1*, whereas *in2* goes through the pass gate P4 – N4 to exit by *out2*. When $c = 1$, the paths cross: *in1* is send to *out2* whereas *in2* is send to *out1*. Thus the straight paths are balanced together, as well as the

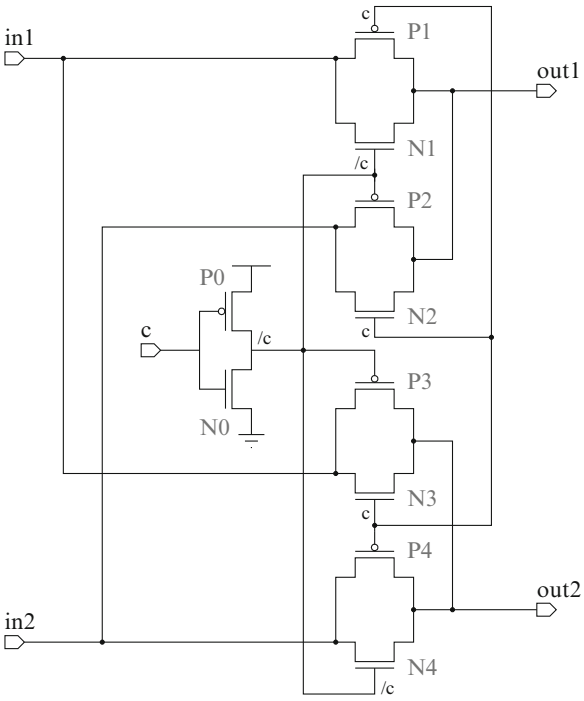


Fig. 7.4 Schematic view of a switch box

Table 7.1 Transistors parameters for the switch box

Transistors	Width (nm)	Length (nm)	Number of fingers
P0	560	60	2
N0	400	60	2
P1, P2, P3, P4	280	60	1
N1, N2, N3, N4	200	60	1

crossed paths. The layout view of the switch box is shown Fig. 7.5. All transistors are placed to obtain a perfect symmetry. The only element of asymmetry is due to the Metal 2 interconnections. However, the impact of this asymmetry is assessed negligible considering the dimensions involved.

7.4 Design of an Arbiter

The role of the arbiter is to determine if the rising edge arrives first on the path 1 or on the path 2. Therefore, the arbiter has to be designed in such a way that no bias is introduced between the two paths. The design must be balanced so that none of the input has an advantage over the other one. A SR-latch built with two

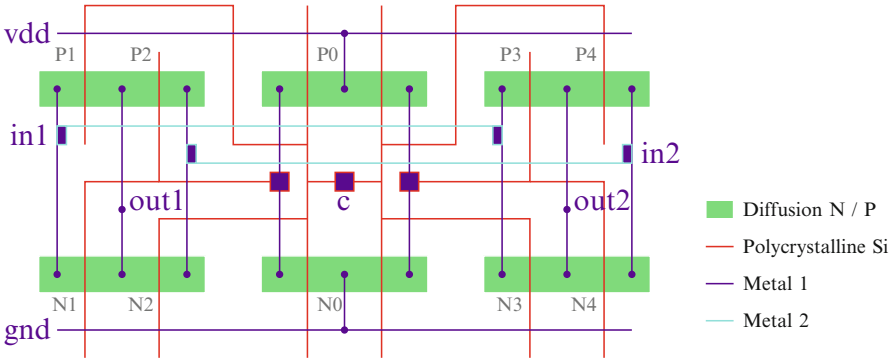
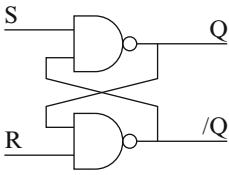


Fig. 7.5 Layout view of a switch box

Fig. 7.6 Symbolic view of an arbiter



coupled NAND2 gates, as shown Fig. 7.6, is the easiest way to design a balanced arbiter. Initially, both inputs are at logical low level, since no rising edge has been propagated yet. Both values Q and /Q are then constraint at logical high level. The SR-latch is in a state which is logically not allowed, since Q and /Q aren't complemented. Yet, this state is stable.

- If the rising edge arrives first on input S, the upper NAND gate switches to logical level '0', since S and /Q are at logical level '1'. The output is then set at level '0'.
- On the contrary, if the edges arrives first on input R, /Q switches to logical level '0'. The output is then set at level '1'.

In both cases, the latch is in a stable state, even after the rising edge arrives eventually at the other input. Resetting the arbiter is done when the enable signal is set back to '0'. Once the falling edge is propagated through the PUF, all nodes are in their initial state. The schematic view Fig. 7.7 is based on the NAND2 standard cell. The parameters are set at their default values:

- PMOS transistors: w = 280 nm, l = 60 nm
- NMOS transistors: w = 200 nm, l = 60 nm

The complemented value /Q is shown on the schematic, but is however not considered as an output of the arbiter. Figure 7.8 shows the layout of the arbiter. The organization of the transistors allows to keep a symmetrical design between the two inputs S and R. The only asymmetry in the layout is the position of the two contacts M1 – Polycrystalline silicon. Given the dimensions, this difference is considered negligible.

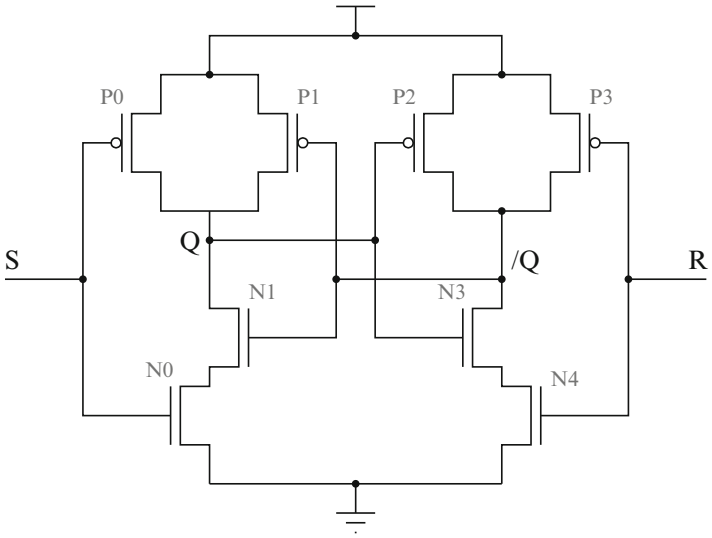


Fig. 7.7 Schematic view of an arbiter

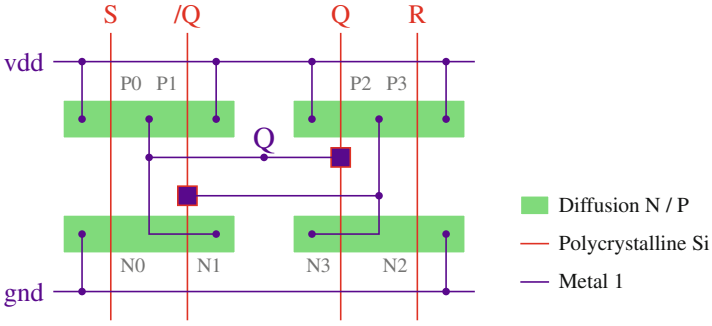


Fig. 7.8 Layout of an arbiter

7.4.1 Design of the Arbiter-PUF

Once the switch box and the arbiter are designed, these cells have to be assembled into an arbiter-PUF. Considering the design of both elementary cells, we opt for a column organization: the first switch box being on the top, the arbiter on the bottom. We choose to design an arbiter-PUF featuring a 16-bits challenge. Sixteen arbiter-PUFs are assembled into the same block, so the response of this IP is 16 bits.

An input shift register and an output shift register are added so data can be serialized. Thus, the number of IOs is considerably reduced:

- Inputs: CLK, RN (for registers), input challenge (serialized), enable, register mode.
- Outputs: PUF response (serialized), output CLK

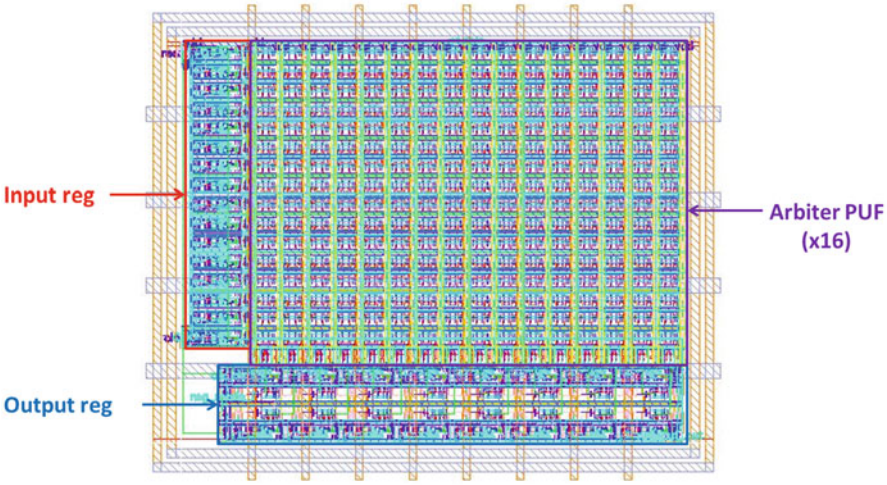


Fig. 7.9 Arbiter-PUF layout

The layout of the arbiter-PUF is finished by integrating power and ground rings which are eventually connected to the VDD and GND pads. The complete layout of the arbiter-PUF is shown Fig. 7.9.

Conclusions

This chapter provides details of the implementation of an arbiter-PUF. It also highlights the difficulties that can be encountered during the design, place and route phases. The particular care that has to be taken to design an arbiter PUF based on switch boxes makes the design more difficult than expected. First, due to strong area optimizations in standard cells, propagation delays in those cells are not balanced. This may introduce a bias in the PUF and it is then preferable to manually place and route each transistor to obtain symmetric data paths. When assembling the cells, the metal layers have also to observe symmetry. Thus, designing an arbiter PUF is not an easy task, and has to comply with many constraints. The main ones are reminded here:

- Design the cells according to the standard cell framework
- Place transistors/standard cells so that active areas (diffusion, N+, P+, etc.) form symmetrical patterns for both datapaths
- Route cells in order to keep this symmetry intact

These constraints are not easy to comply with, even if the design is simple. The sub-micron technologies have strong and numerous design rules, making also the design not transferable to another technology. As a conclusion,

(continued)

a reflection is needed regarding the cost – in terms of human and financial resources – of designing an arbiter-PUF based on switch boxes. Considering the complexity of the task, other methods may be preferable. Designing PUFs with less place and route constraints – such as the loop-PUF described in Chap. 6 – is definitely a valuable approach.

Reference

1. Gassend, B., Clarke, D., Van Dijk, M., Devadas, S.: Silicon physical random functions. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, pp. 148–160. ACM (2002)

Chapter 8
Secure Key Generator Using a Loop-PUF

Julien Francq and Geoffrey Parlier

Abstract In this chapter, we describe a secret key generator which uses a Physically Unclonable Function: the Loop-PUF (LPUF). After a short description of this latter, we will describe the concept and the architecture of our secure key generator (PUFKY). We will detail all its components and all the security analysis we made. We will finally show that the LPUF, if its location is well chosen, has sufficient properties to be used as a robust and secure key generator. More generally, this chapter will allow the reader to implement a PUFKY quickly from scratch.

8.1 Introduction

Physical Unclonable Functions (PUFs) [2, 5, 8] are adapted to be used as hardware key generators because they allow to avoid a system to store sensitive informations (i.e. cryptographic keys) by regenerating them at each usage. This protects this kind of key generator against *reverse engineering* attacks.

Secure boot applications can take advantage of PUF usage. A secure boot allow to run only authenticated codes (i.e. boot loaders) on a platform. It is thus a protection against malwares because it prevents from the loading of non-authorised binary software during the boot of the component. For example, computers including a secure boot will not operate non-authorized operating systems.

A PUF allows to extract a secret key directly from physical parameters of the circuit. Thus, if we combine the secure boot approach with a PUF, we can obtain a high-level security platform offering a protection against counterfeiting and cloning. However, before generating a key, we must post-process the PUF response to make it reliable with an efficient Error-Correcting Code (ECC).

This chapter describes all the steps necessary to implement a PUFKY using a PUF. The PUF used in this work is the Loop-PUF (LPUF) which has shown to have good security properties. It will appear at the end of our study that the LPUF is sufficiently secure to be used for a robust and secure key generator.

J. Francq (✉) • G. Parlier
Airbus Defence and Space – CyberSecurity, Bd Jean Moulin, CS 40001,
MetaPole, 78996 Elancourt Cedex, France
e-mail: julien.francq@cassidian.com; geoffrey.parlier@cassidian.com

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_8

After a short description of the LPUF principle, it will be described the concept and the architecture of our secure key generator (PUFKY). In another section, we will also detail all the security analysis we made on our architecture, before summarizing our results and concluding.

More generally, this chapter will allow the reader to implement a PUFKY quickly from scratch.

8.2 LPUF: Architecture and Properties

LPUF is a PUF from the Delay-Based PUF family. The fundamental principle of such kind of "delay-based PUFs" is to compare a pair of structurally identical/symmetric circuit elements and measure any delay mismatch introduced by the manufacturing process variations. A LPUF is a sort of mix of an arbiter PUF and ring-oscillator PUF (see Fig. 8.1).

The architecture of this PUF is very simple and easy to implement on a FPGA. It is made of N controlled delay chains which are serially gathered. All these chains form a big oscillation loop. Like for *Ring Oscillator PUF* (ROPUF), this latter allows to measure the number of oscillations (number of periods) in the loop during a fixed time window. Moreover, each delay chain is identical and is made of M delay elements all connected together. Its dimensions depend on the parameters M and N and its simple design allows a place-and-route of the circuit in FPGAs without particular constraint. The measurement of the response of this PUF is made thanks to a system of comparisons between the different frequency measurements computed for one challenge. The creation of an entity (LPUF IP) on a FPGA is not complicated because the design is sufficiently scalable and flexible. Using only one oscillation loop decreases significantly the noise on a PUF response. Finally, each of these delay elements does not need Switch Blocks. It is not needed to cross the signals, that eases greatly its implementation on FPGA. The only constraint concerns the fact that the delay elements must follow themselves physically on the FPGA (see Fig. 8.2).

LPUF works like the other PUFs: it receives on its inputs one challenge, this latter is made of N words $(C_1 C_2 \dots C_N)$, each has a size of M bits. Thus each chain receives a word C_i of M bits. That also means that a challenge is made of $N \times M$ bits. To communicate the bit value coming from the challenge at each delay element, the LPUF has a controller (the LPUF Controller) which must also send

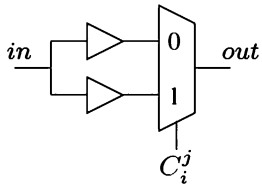


Fig. 8.1 One element of a LPUF

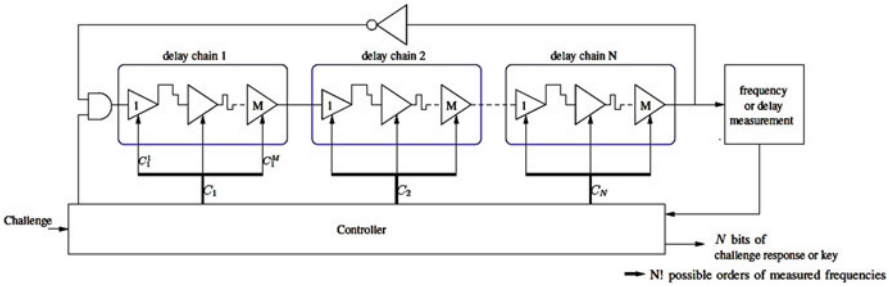


Fig. 8.2 LPUF architecture

Table 8.1 LPUF vs. Arbiter PUF – FPGA Cyclone II

Performance metrics	Arbiter PUF (%)	LPUF (%)
Randomness	0	≈100
Uniqueness	97.73	95
Steadiness	99.07	98.7

back the binary response. The LPUF controller allows also to manage the good distribution of the challenge as well as the recovery of the measured values and the conversion in binary response, which represents the signature of the PUF defining the integrated circuit, here the FPGA.

The window time for the measurements serves as a reference to compute the number of oscillations in the loop of the LPUF. This value is linked to the LPUF frequency. We remark that, the bigger the time window is, the more reliable is this measurement, because it increases the measurement time and then the number of gathered delays.

The LPUF has been already tested on a ALTERA FPGA (Cyclone II). Table 8.1 compares the results obtained with a LPUF and an equivalent Arbiter PUF.

LPUF provides more randomness by far, so that justifies its choice for key generation.

To modify these results, we can choose different value for M and N which defines physically this PUF.

In summary, the LPUF, thanks to its design offering a big flexibility with its parameters and a easy implementation, allows to obtain a low-cost mean in terms of resources and clock cycles to generate one source of secret random data. Its performance criteria show that its usage for the key generation is appropriate compared to other PUFs. This is a basic requirement for a secure boot application.

However, the performance criteria Steadiness indicates that, like for other PUFs, a challenge-response couple is not directly usable because it is not enough stable and does not give sufficient information. Some pre-processing steps will be then necessary (computed by the Challenge Controller and the Lehmer Gray component, see resp. Sects. 8.4.1.4 and 8.4.1.6). A post-processing step (Error Correcting Code) will also be needed to decrease the error rate of the response of the LPUF (see Sect. 8.5.2).

8.3 Secure Key Generator Based on the Loop-PUF

When a PUF is used to generate a cryptographic key, it has to produce randomly distributed random data with a high entropy. The PUF must also have a robust (reliable) behavior when used in this kind of purpose. However, a PUF is not 100 % reliable by nature, so a post-processing of the PUF responses is needed before using this kind of random data for generating a cryptographic key. Moreover, this key generation phase must be as lightweight as possible and resistant to some attacks like *Machine Learning* and *Side-Channels*.

We call a *Controlled PUF* (CPUF) a system which contains the PUF with the modules needed to manage it. Our goal is to drive our CPUF with a API (*Application Programming Interface*) which allows to limit its access.

A CPUF is made of two elements: a PUF and a control logic block which contains an error correcting code to correct the responses, followed by a cryptographic hash function to counteract modelization attacks. With these blocks, the response of a PUF can be used as a cryptographic key; we call this procedure KEY EXTRACTOR.

8.3.1 PUFKY: Concept

It is possible to use a CPUF as a PUF-based Key Generator (PUFKY). A PUFKY is made of a PUF and a *KEY EXTRACTOR* (or *Fuzzy Extractor*). The extraction of a key with a *Fuzzy Extractor* can be done in two steps. The first step is called "generation step". It is done only once in the life cycle of the component. It allows generating responses from the PUF and extracting a key as well as additional information called *Helper Data*. The second step allows to reconstruct at any time the key obtained in the generation step. To do so, it is needed to implement Error Correcting Code which needs the *Helper Data* to reconstruct the key. Then, the PUFKY concept provides a key without storing it in non-volatile memory. Only the Helper Data is stored in such memory and the data it contains are not critical if they are public, or retrieved with invasive means.

The PUFKY must successfully do two important tasks: increase the reliability of the PUF response (which is noisy by nature), and get a sufficient entropy. This is why we use a *Fuzzy Extractor*, based on error correcting code technology. This PUFKY concept has been first proposed by Leuven University researchers [4, 7]. It is the starting point of our work.

8.3.2 PUFKY: Architecture

The global architecture of a PUFKY is depicted in Fig. 8.3. It implements the LPUF developed in the ENIAC funded TOISE project.

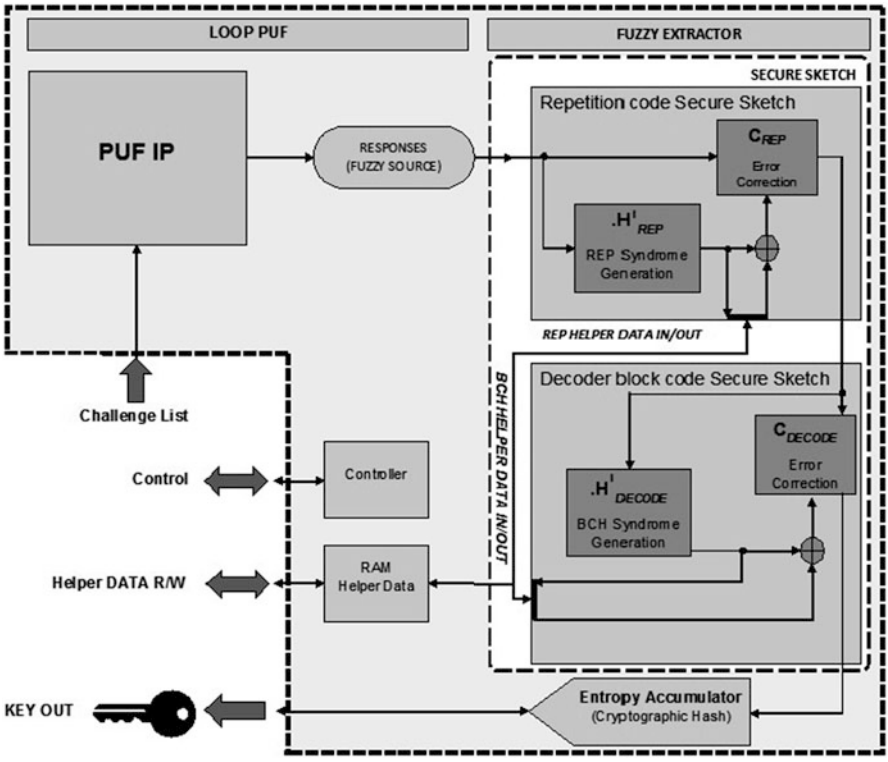


Fig. 8.3 PUFKY architecture

To correct the noisy response of the PUFKY, our *Fuzzy Extractor* uses an error-correcting code which works in the two steps "Generation" and "Reproduction". The *Secure Sketch* is made of an encoder/decoder block $C(n, k)$. The best way to proceed is to use the decoder to generate syndromes and an Error-Correcting Code (ECC). The final choice of this ECC depends on the quality of the PUF responses.

In Fig. 8.3, the "*Helper Data*" interface allows to write or read data, depending if we are in Generation or Reproduction Phase. Moreover, the repetition block (majority vote) of the ECC allows to increase the reliability of the LPUF response, but increases the global computation time since we repeat many times the same measurement. Finally, *Helper Data* do not have an influence neither on the response quality, nor on the final entropy. It only allows to store the syndrome used in the reproduction phase, as it will be explained in the following. We must just ensure that the public information leaked by helper data is not sufficient to reconstruct the key (see Sect. 8.3.3.4).

In summary, PUFKY implements these modules:

- The ***LPUF***
- A ***Fuzzy Extractor*** comprising:

  - An ECC (*Secure Sketch*) which uses a correction with syndromes.
  - A cryptographic hash function ***Entropy Accumulator***.
- The controller of the PUFKY which is in charge of the key generation.
- A communication interface.

8.3.3 Key Extractor

This kind of information extractor is different that standard ones. It has to collect information from the PUF which are not uniform, noisy and with a low entropy. This source of information is called *Fuzzy*. The extraction method of the *Fuzzy Extractor* is specific because of the *Secure Sketch* procedure.

8.3.3.1 Fuzzy Extractor

Here are described in more details and in Fig. 8.4 the two steps of the *Fuzzy Extractor* (**Generation** and **Reproduction**):

- Generation step corresponds to the enrollment phase of the PUF. It is done only once in the life cycle of the component. It is the time where the couple *Challenge-Response* (CRP) is generated. This phase comprises also the generation phase of the *Secure Sketch* which produces at the same time the original cryptographic key and the Helper Data.
- Reproduction step is one of the *Secure Sketch* phase where the PUF response is generated once again with the same challenge. This step allows to reconstruct at any time the key obtained in the generation step. However, because the response has slightly differences with the one generated in generated phase, the use of the

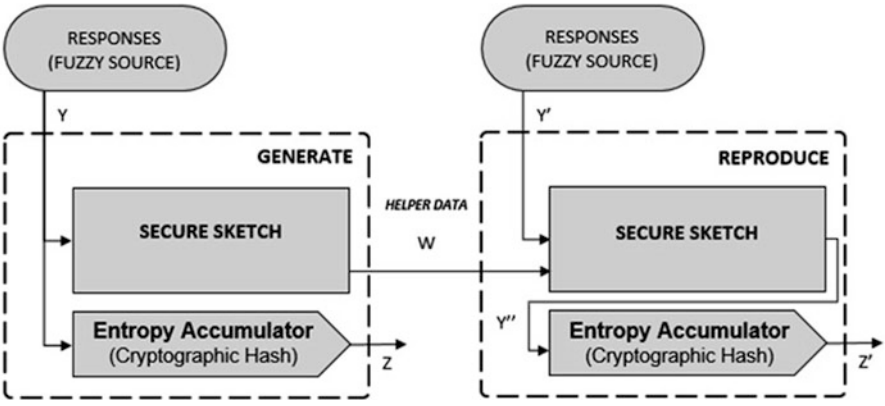


Fig. 8.4 The two steps of the *Fuzzy Extractor*

Secure Sketch in reproduction step corrects the PUF response using the created Helper Data, and sends it in the *Entropy Accumulator* which reconstructs the key. The correction is then done with the decoder block of the ECC and the Helper Data.

8.3.3.2 Secure Sketch

Secure Sketch allows reproducing a signal using a noisy source of data. It transforms a low entropy source in a reliable information with high entropy. To do so, it comprises an ECC which computes a syndrome. This latter is an information reused in reproduction phase for reconstructing the correct value of the data. Our ECC block, $C(n,k)$ can be a cyclic/linear one. This kind of ECC uses a matrix $\mathbf{H}$ called *Parity-Check* matrix and a generation matrix $\mathbf{G}$. These two matrices are defined using the relation: $G \times H^T = 0$. The *Secure Sketch* is a pair of two procedures, and in the following our response matrices are denoted y, and y'.

- During the generation step, the *Secure Sketch* produces $(n-k)$ bits which are stored in *Helper Data* matrix w:

$$w := y \times H^T. \tag{8.1}$$

- During the reproduction phase, the *Secure Sketch* computed the syndrome S of the ECC $C(n,k)$, such that:

$$S := y' \times H^T \oplus w \equiv (y \oplus y') \times H^T \tag{8.2}$$

- Then, the ECC block decodes the information, such that:

$$S := e \times H^T \tag{8.3}$$

The e vector has a low *Hamming Weight* (HW) and a low correction rate t, such that $HW(e) \leq t$. It allows to reproduce the new corrected PUF output y'', such that:

$$y'' := y' \oplus e \tag{8.4}$$

- The validity condition of the corrected response is gauged by the *Correctness* criteria. It allows to check if the reproduction of the computed value corresponds to the scheduled one. This condition can be expressed as follows:

$$\text{If } HW\left\{(y;y')\right\} \leq t, \text{ then } e := y \oplus y', \text{ thus } y'' \equiv y \tag{8.5}$$

This procedure explicits that the matrix w which corresponds to the Helper Data must be stored during all the life cycle of the entity PUFKY since it is the link

between the generation phase and the reproduction phase. Data of w corresponding to the parity bits of the syndrome are public. The error correcting rate t of the *Secure Sketch* must be determined during the development phase of the PUFKY. Theoretically, it must be as low as possible but it must also be sufficient for the key generation. It is then linked to the quality of the source and then to the physical parameters of the Loop PUF.

8.3.3.3 Entropy Accumulator

The *Entropy Accumulator* can be a cryptographic hash function. In our work, the implemented hash function is SHA-3. This latter uses the output of the *Secure Sketch* which corresponds to the corrected output of the PUF to process a functional cryptographic key. A hash function is injective: It takes an input message of arbitrary length and produces a fixed length digest (for our SHA-3 implementation, it is equal to 256 bits). We obtain at the output of the hash a character chain, which corresponds to the digest of the PUF responses. Hash function breaks the link between the challenge of the LPUF and the produced key, which will avoid all the kind of *Machine Learning* attacks.

8.3.3.4 Remarks on the Security of the Generator

If we want a secure key from the PUFKY, the final entropy of the binary chain at the input of the Entropy Accumulator must be equal or greater than the wanted key size. That means that the remaining number of independent bits must be equal *a minima* to the key size. In our case, we want an entropy equal or greater than 256 bits.

Let L be the size of the PUF response, and N the number of responses coming from N challenges to generate a key. Let w be the matrix containing the parity bits of our syndromes coming from the ECC $C(n, k)$, which has the size:

$$dim\{w\} = dim\{N \times (n-k)\} \tag{8.6}$$

Let ρ be the entropy rate of the PUF response (ρ is determined through some tests, it is a sort of "security coefficient"). Then, the security equation that the LPUF must obey is:

$$[\,\rho \times dim\{N \times L\}\,] - dim\{N \times (n-k)\} \geq dim\{KEY\} \tag{8.7}$$

This criteria is critical since it guarantees that the public information coming from the parity bits is not sufficient to reconstruct the key by an attacker, since the final number of independent bits is greater than the key itself.

Fig. 8.5 Test-bench

8.4 LPUF: Study

8.4.1 Test-Bench Presentation

As aforementioned, the LPUF IP has been developed during ENIAC funded TOISE project: VHDL sources and one *Hard Macro* (HM) have been delivered which corresponds to the *design* of one LPUF made of 4 chains, each one made of 16 delay elements.

Our test-bench uses a Xilinx ML507 development board containing a FPGA Virtex-5 (XC5VFX70TFFG1136) and one Power-PC 440.

This allows to test easily the LPUF thanks to a basic client implemented in C language which communicates with the board using the serial port. It allows sending challenges and measuring the responses which are related to the oscillation frequencies of the LPUF for given challenges.

Figure 8.5 depicts the test-bench.

8.4.1.1 IP PUF Specifications

This IP is made of a PUF module duplicated N times and a controller followed by an interface for writing and reading in the registers.

In the initial package [1] we have 36 HMs implementing the LPUF.

Each HM is made of two different PUF delay elements. This element contains two identical buffers and a multiplexer. The use of these buffers is needed to increase the delay (and then decrease the oscillation frequency to measure it). To select between these two kinds of PUF, we can use a control signal.

The two kinds of PUF which are present in one HM are (see Fig. 8.6):

- *Arbiter PUF*,
- LPUF.

In both cases, the same delay chains are used for both PUFs. But for the LPUF case, these latter are serial. Finally, one inverter is added between the chains to allow the PUF to oscillate. The output of the loop is called *outlpuf*: it allows to compute the oscillation frequency of the LPUF, but this output does not correspond to the final response of the PUF. A post-processing (encoding) step is needed to use the frequencies and finally produce this final response (we will implement Lehmer-Gray encoding, see Sect. 8.4.1.6).

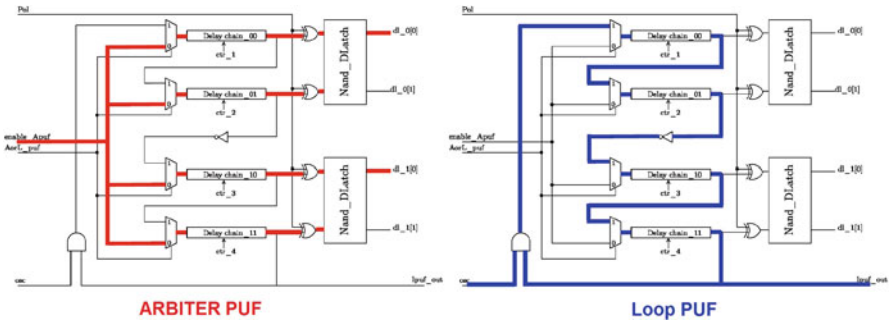


Fig. 8.6 Arbiter PUF vs. loop PUF

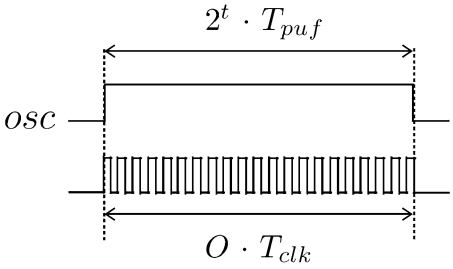


Fig. 8.7 Measurement window of the LPUF

To push the tests further on the ML507 board, we raised the number of PUF modules to 132 HMs, respecting place-and-route constraints of our HM which occupies 24 slices in the FPGA.

The PUF period T_{puf} is measured by counting the number of periods O of the clock clk during an oscillation window $= 2^t \times T_{puf}$ (see Fig. 8.7).

t obeys to the following relation:

$$t \leq 16 + log_2(\frac{F_{puf}}{F_{clk}}) \tag{8.8}$$

In our measurements, we could see that for $F_{clk} = 25$ MHz, we obtain thanks to the O value that $F_{puf} = 100$ MHz. We can then conclude that $t_{max} = 18$.

The HM of the LPUF (see Fig. 8.8) needs a controller to work.

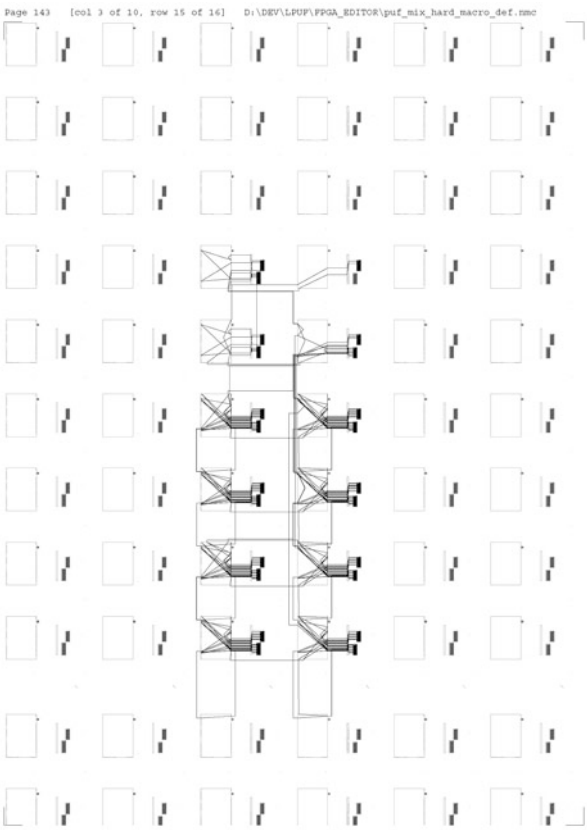
The HM also has some control signals to define its mode and also 4 16-bit input signals. These signals are the challenge itself which defines the value of the multiplexers of the 4 delay element chains. Finally, we got at the output the oscillation signal of the PUF.

Figure 8.9 depicts the place-and-route of the design on the ML507 board.

We detail hereafter the meaning of the colors:

- Violet: PLL 25 MHz,
- Yellow: Frequency FIFO,
- Green: OSC signals,

Fig. 8.8 Loop PUF HM



- Orange: LPUF controller,
- Cyan: Interface PUF,
- Red: UART IP.

Figure 8.9 allows to see that the area on the right of the Power PC is made of IPs which allow to communicate with the LPUFs. We can find there the UART IP, one FIFO playing the role of frequency adaptation between the IPs and the I/O interface. There is always activity in this area when a PUF is working. The proximity with some HMs can perturbate this one (see Sect. 8.4.2.4).

8.4.1.2 Client Specifications

Once the LPUF is implemented, we have decided to study its performance indicators but for doing this, we got to implement a client which integrates a new chain of processes such that the total permutation of challenges, read/write data in files, etc. to realize a post-processing step during the exploitation of the results.

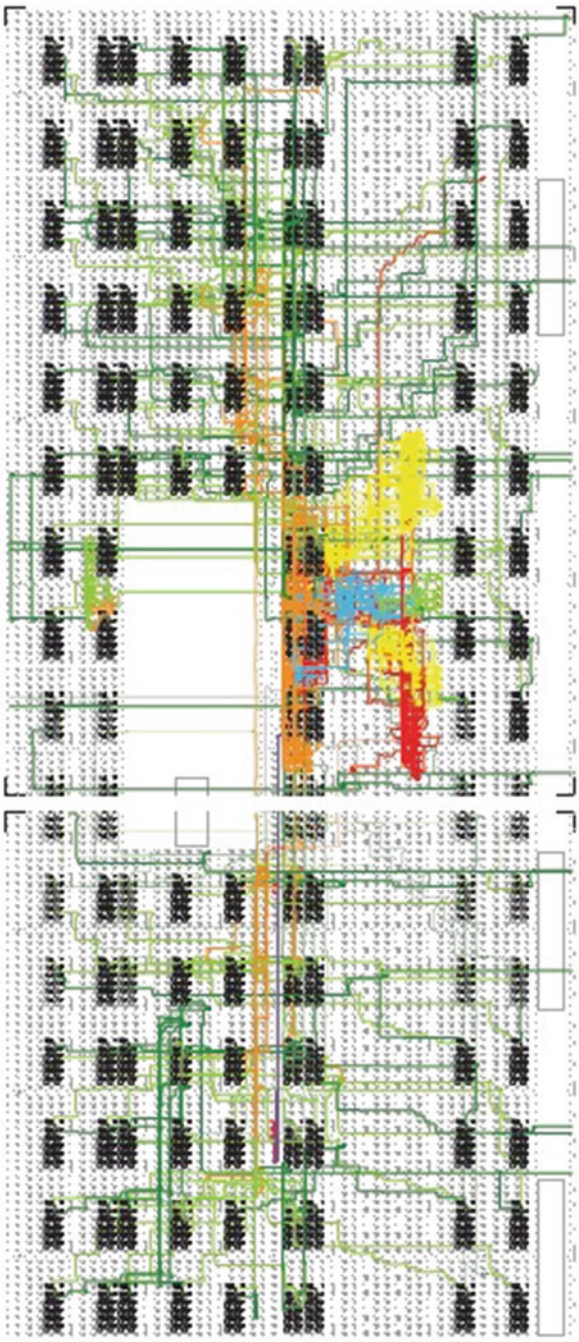


Fig. 8.9 Place-and-route of the design on ML507

Here is a list of functionalities implemented in the client:

- Get a list of challenges from a input file.
- Function which computes the needed 24 permutations of one challenge.
- Function which sends these 24 sub-challenges to a PUF.
- Modify the function which sends a challenge to a PUF and sends back the number of oscillations.
- Include the management of client parameters and the interface (parameters of UART, number of PUFs, number of iterations T, number of challenges).
- Management of the folder names, and savings of the data in CSV format.

Once the data saved, they can then be processed with software like Matlab or its open-source equivalent Octave.

8.4.1.3 Choice of the Identification Method

The identification step is really important because it corresponds to our signature step of the PUF. As well as the RO-PUFs, oscillation values must be transformed in a binary chain.

The identification method must guarantee the following points:

- To get a sufficient quantity of random information to generate a key which is at least 256 bits long,
- Get a uniform random source,
- Get a response with a high entropy,
- Use a light process for the FPGA.

All the properties of our test-bench have been determined thanks to our Hori method [3] implementation developed on Octave.

8.4.1.4 Challenge Controller

This entity allows to manage the control words which will be sent at the input of the LPUF. One challenge is in fact 4 16-bit words $C_1C_2C_3C_4$. Each word C_i is sent to a LPUF control chain. We can create new challenges by permuting the order of the words. For example, the first permutation can give: $C_1C_2C_4C_3$. Globally, there exists 24 permutations, this allows to create 24 *sub-challenges* coming from one challenge. The architecture of the challenge controller is depicted in Fig. 8.10.

Identification method of the response is then computed with the aim of 24 frequencies written on 16 bits, which offers a big quantity of information.

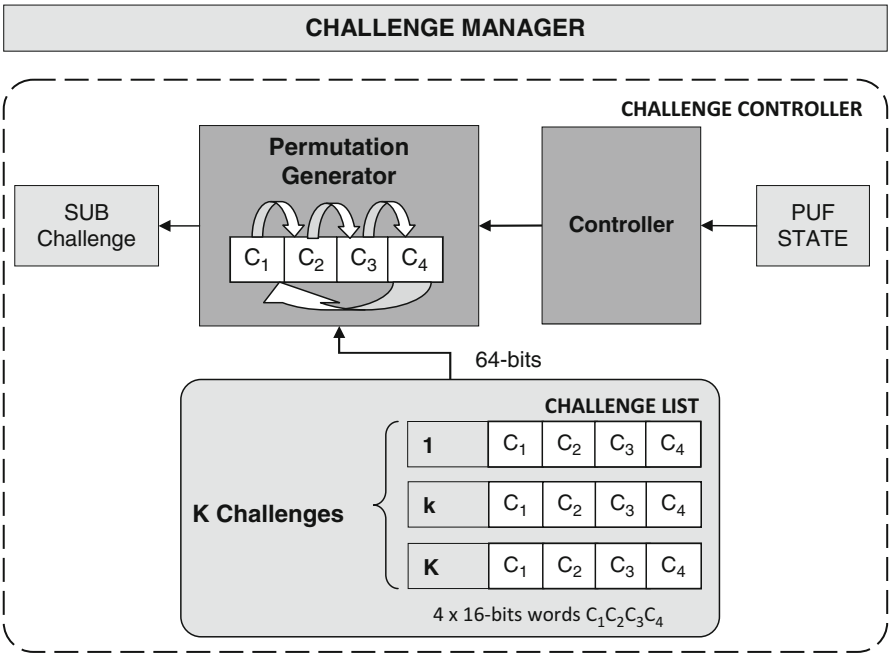


Fig. 8.10 Architecture of the *Challenge Controller*

8.4.1.5 Direct Pair Comparisons

This method consists in studying the sign of the difference between two measured oscillation frequencies. Only one response bit r is generated depending on the result of the difference; positive, equal or negative. We obtain the following rule:

$$\begin{aligned} &\forall \ \left\{F(i), F(i')\right\} \in \mathbb{F}\,[\mathbb{I}] \ \text{ with } i \neq i' \text{ such that } (i, i') \in [0; 23] \\ &\text{If } \left[F(i) - F(i')\right] \geq 0 \text{ then } r = 1, \text{ else } r = 0 \end{aligned} \tag{8.9}$$

This comparison is not too much costly. Moreover, it offers a big number of possible combinations ($C_2^{24} = 276$). Response length can be thus written on 276 bits with this method, but results with some challenges show that the binary response chain possesses a low entropy and that it is noisy after some iterations. Some frequency values are too close, then it modifies the sign of the comparison at some moments and so the response. Finally, our test tools show that a majority of the bits are redundant ones. After a study on 272 concatenated bits of a response, our tool exhibits an entropy of 4.27 bits for a byte (entropy rate: 54 %). Moreover, we can remark that the bits repartition in a response is cyclic and deterministic: only the 24 first combinations are really independent because their results give information on the ranking order of the measured values, and so a probability on the following ones. It is then possible to determine the rest of the response in a probabilistic way.

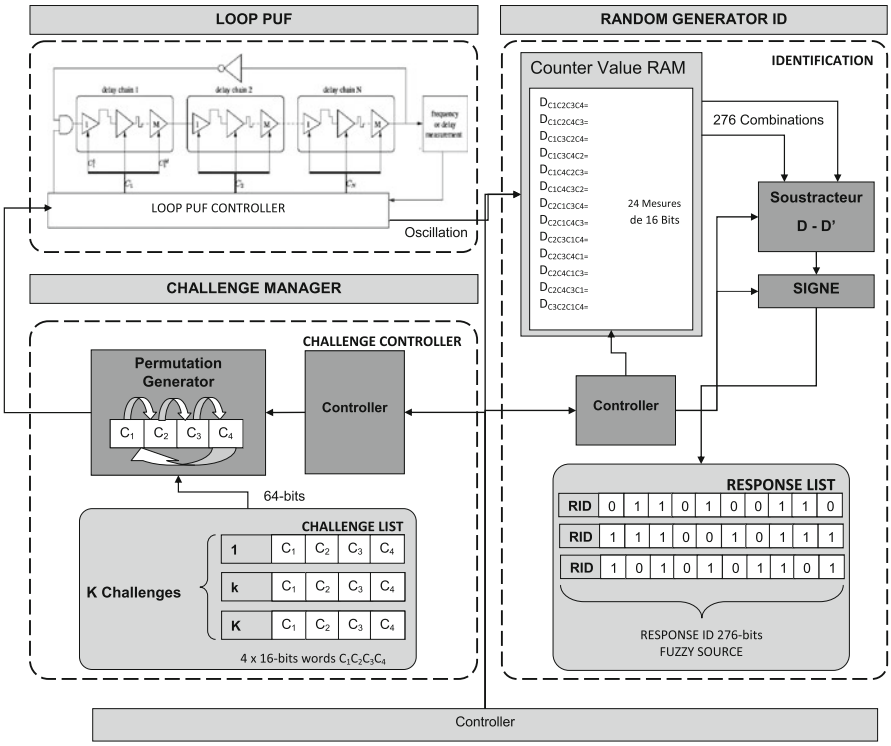


Fig. 8.11 Direct pair comparisons architecture

We can then conclude that for a 276-bit response, only the first 24 ones are really 100 % independent because each bit comes from at least one new measurement on the 24. For a cryptographic key generator, a noise response of 24 bits is insufficient. To overcome this problem, it would be needed to use a big number of challenges, which will imply an increasing measurement time and a more noisy response. Thus, the pairs' comparison (depicted in Fig. 8.11) has not been chosen for our secret key generator.

8.4.1.6 Lehmer-Gray Encoder

Lehmer-Gray Encoder, created by Derrick Lehmer (a twentieth century mathematician) is a sorting algorithm which allows to calculate a unique numeric vector featuring a permutation order of n measured frequencies. This vector then allows to calculate Lehmer coefficients which are the responses of the LPUF. A big advantage of this method is that it does not need to select a comparison order to calculate the coefficients. This offers a more effective approach than the pairs' comparison which needs to compute all the possible combinations, or select a pre-determined order. Lehmer code is then more robust to mathematical modelization.

Lehmer code uses a frequency vector with a dimension equal to 24. We have then a finite set of $n = 24$ frequencies, $\mathbb{F}[24]$ such that $f^n = (f_1, \ldots, f_n)$. The Lehmer encoder allows to calculate the vector with the order $r^{n-1} = (r_1, \ldots, r_{n-1})$ with $r_i \in \{0, 1, \ldots, i\}$. Thus, there's $n!$ possible Lehmer vectors for a given frequency vector f^n. Lehmer vectorial representation is then unique for a set of 24 measurements corresponding to only one challenge of the LPUF. Lehmer vector corresponds to a LPUF signature, allowing the final computation of the response.

Lehmer coefficients are thus computed with a vector f^n such that:

$$r_j = \sum_{i=1}^{j} gt(f_{j+1}, f_i{}') \text{ with } gt(x, y) = 1 \text{ if } x > y, \text{ else } 0.$$

We have chosen to use the function $gt(x, y)$ as a comparison tool. It is more reliable than the study of the sign at the subtractor output since the measured LPUF frequencies are the result of oscillations measurements coming from the delays accumulated by the LPUF chains. Cherif et al. [1] has shown that the LPUF output coming from an oscillation is linear with temperature and power supply variations. We can deduce that the superiority comparison method between two measurements has more chance to be more stable than with a sign study. Thus, it brings reliability to the response and allows to minimize noise generated by the PUF itself.

Lehmer code possesses thus interesting properties to reduce LPUF noise: we can observe that if the result of a comparison between two values changes because of a perturbation generated by the LPUF, then the Lehmer coefficient will vary of only ± 1.

Once the 23 Lehmer coefficients are computed, information of this vector must be transmitted in one binary chain, where each bit must be useful. We must then compress information. For this, coefficients are encoded under Gray binary format which makes the response particularly reliable and robust to the noise of the LPUF.

Knowing that the coefficients $r_i \in \{0, 1, \ldots, i\}$, they are thus bounded, so the allocated space in the binary chain varies regarding the coefficient. Coefficients are then encoded on 2, 3, 4, 5 bits maximum, regarding the number of bits that can take the maximum value of the coefficient. A LPUF response for a 64-bit challenge is then 88-bit long, to be a multiple of a byte (see Fig. 8.12).

Applying this method, we obtain a new architecture of the PUF, showed in Fig. 8.13.

8.4.2 Experimental Results

In the following, we analyze the properties of our developed LPUF IP.

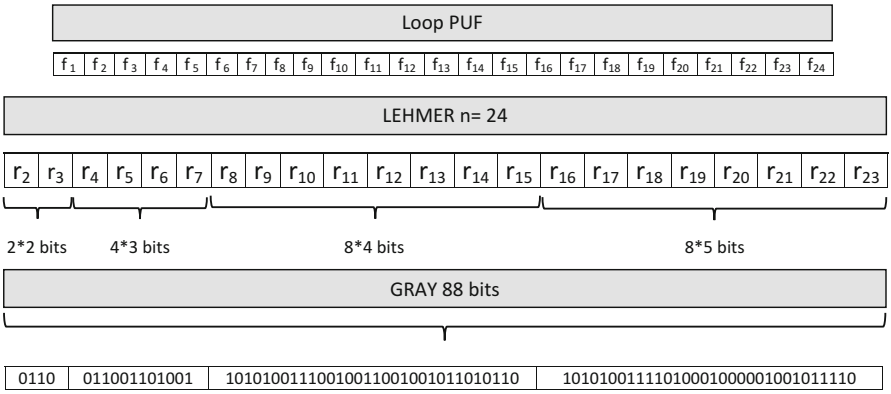


Fig. 8.12 Lehmer Gray encoder

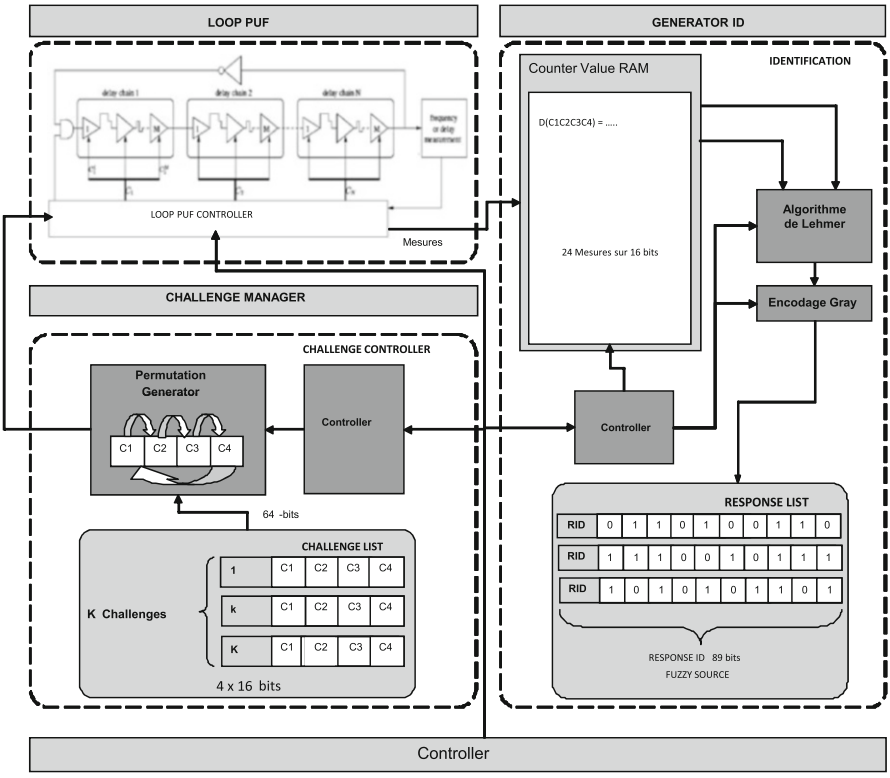


Fig. 8.13 IP PUF architecture

Table 8.2 Area occupied on ML507 board

Module	Occupied slices	FPGA usage (%)
132 LPUFs	3,168	28
Others	384	3
Total @ 10 MHz	3,552	31

Table 8.3 Parameters

Parameter	Value
LPUF frequency	10 MHz
Signal t	16
Time window	260 µs
N	132 LPUFs
K	20 challenges
T	20 iterations
L	88 bits

Table 8.4 Duration

Step	Time (h)
132 LPUF measurements	4
Lehmer Gray scripts	1
Hori scripts	9
Total @ $t = 16$	15

Table 8.5 Experimental results with 132 LPUFs

Criteria		Average	Standard deviation	Interval	Confidence: 95 %
Probability	p	*0.4631*	*0.0080*	[*0.4618*	*0.4645*]
Randomness	H	0.8975	0.0215	[0.8937	0.9011]
Steadiness	S	0.9590	0.0065	[0.9579	0.9601]
Correctness	C	0.9530	0.0073	[0.9517	0.9542]
Diffuseness	D	0.8905	0.0085	[0.8890	0.8920]
Uniqueness	U	0.6472	0.0601	[0.6368	0.6575]

8.4.2.1 Analysis on 132 LPUFs

In this experience, design properties and parameters are as follows (Tables 8.2–8.4):

Table 8.5 summarizes the results obtained with our architecture of the LPUF.

We can notice that the entropy rate is around 90 % in average, data from response are then usable for key generation. Moreover, each of these results has a low standard deviation, we get then a response which has a stable behavior as a function of time, challenges and location in the FPGA. The second important indicator is the steadiness of the response (≈95 %), showing its capacity to resist to environmental perturbations on the PUF. For 20 iterations, the correctness of the response is in average equal to 95 %, which means that at each iteration, around only 5 bits of the LPUF IP response vary in average. Finally, we can see that the diffuseness (number of different response bits result of two challenges) is equal to 89 %. Moreover,

each response, for a given challenge and LPUF, is unique since in average 65 % of the response bits are different between two LPUFs. The new improved LPUF IP has then the behavior of a secure Physically Unclonable Function.

We also remark that the context of the experience is important, because the measurement set is spread on one experience which has a 4 h duration. We can then deduce that the use of the responses for key generation will not pose problems at one time. However, despite acceptable results for a usage in a *secure boot*, these results comes from 132 LPUFs and 20 one after the other iterations. It is thus interesting to see how behaves the LPUF spatially in the FPGA (see Sect. 8.4.2.4). Finally, this characterization result gives indications on the ways to correct the response which has a 5 bits variation in average. It is very important to choose the most efficient error correcting code, because it must provide a 100 % correction rate for the key generation process (see Sect. 8.5.2).

8.4.2.2 Analysis on 1 LPUF

To push the measurements a little bit more further, a second experience testing the PUF endurance has been set with 200 iterations and 20 challenges. This experience uses only one LPUF but uses a 25 MHz frequency. The time window chosen for the experience is the biggest possible regarding the limits of the system ($t = 18$). An iteration can be then measured in 10 s in average with 20 challenges. Table 8.6 details the results.

This experience shows that the more we re-use some challenges, the more stable is the LPUF response. This advantage can be very useful for key generation. This easy technique (repeating a measurement) is advantageous since it increases the stability (from 95 to 97 %). Thus, there are 3 bits in average that are different after 200 iterations. Moreover, at 25 MHz, the time for a measurement becomes 600 ms, which is still acceptable for a *secure boot*.

8.4.2.3 Inter and Intra Distance Measurements

The first experience (132 LPUFs, 20 iterations, 20 challenges) also allows to trace intra- and inter-distance curves (see Fig. 8.14). This result allows to check that the new architecture respects the properties and definitions of a PUF. And it also allows to compare the LPUF with other PUF types.

Table 8.6 Experimental results on one LPUF

Criteria		Average	Standard deviation
Probability	p	*0.4581*	*0.49*
Randomness	H	0.8839	NC
Steadiness	S	0.9727	0.1207
Correctness	C	0.9685	0.1337
Diffuseness	D	0.8938	0.1781

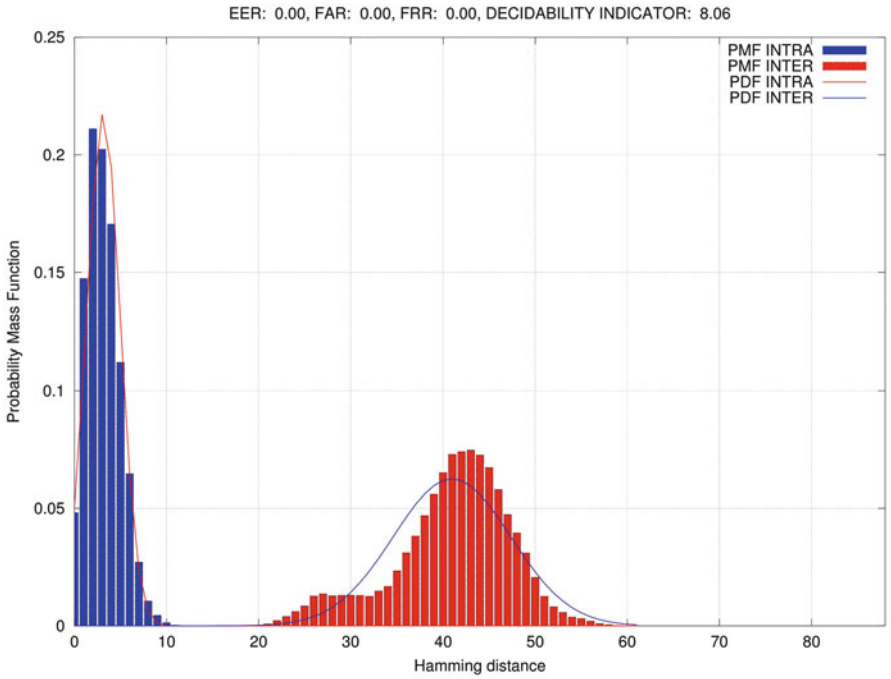


Fig. 8.14 Inter & intra distance of the LPUF

We can first denote that the inter and intra-distance follows a normal distribution with an average intra-distance of 3.5 % and inter-distance of 46.5 %. If we compare with the other PUFs of the literature, we can claim that the LPUF is very competitive.

A second important point is that there is no false positives or negatives because there is no intersection between the intra- and inter-distance curves. Thus, LPUF allows to identify a unique and unclonable response for a given challenge and LPUF.

To better understand and compare the inter- and intra-distance results, a decidability criteria, based on the average of inter- and intra-distance of a LPUF, has been used. This criteria allows to rank the LPUFs: The bigger it is for a LPUF, the more it is efficient (low intra-distance and big inter-distance).

$$dprime = \frac{|\overline{Intra} - \overline{Inter}|}{\sqrt{\frac{Var(Intra)+Var(Inter)}{2}}} \tag{8.10}$$

8.4.2.4 Spatial Analysis of the LPUF on the FPGA

Once the criteria *dprime* is computed for each LPUF, we can spatially represent the performances of the 132 LPUFs implemented in the entire FPGA (see Fig. 8.15).

SLICE X/Y	X4	X12	X16	X24	X32	X40	X44	X64	X72
Y152	124	125	126	127	128	129	130	131	132
Y142	115	116	117	118	119	120	121	122	123
Y132	106	107	108	109	110	111	112	113	114
Y122	97	98	99	100	101	102	103	104	105
Y112	88	89	90	91	92	93	94	95	96
Y102	79	80	81	82	83	84	85	86	87
Y92	73	74	POWER PC			75	76	77	78
Y82	67	68				69	70	71	72
Y72	61	62				63	64	65	66
Y62	55	56				57	58	59	60
Y52	46	47	48	49	50	51	52	53	54
Y42	37	38	39	40	41	42	43	44	45
Y32	28	29	30	31	32	33	34	35	36
Y22	19	20	21	22	23	24	25	26	27
Y12	10	11	12	13	14	15	16	17	18
Y2	1	2	3	4	5	6	7	8	9

Fig. 8.15 Locations of the 132 LPUFs in the FPGA

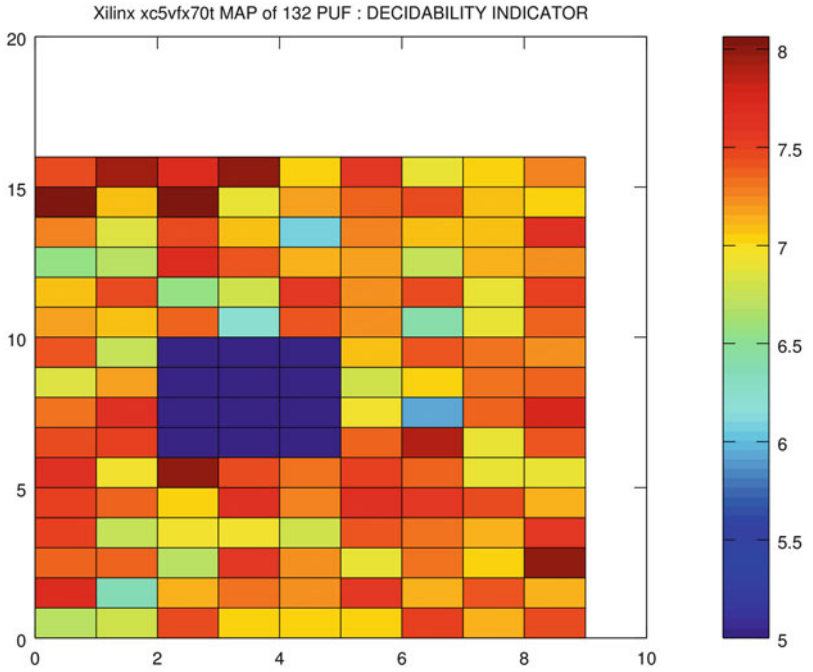


Fig. 8.16 *dprime* values on ML507

The criteria *dprime* varies between 6 and 8, but we can remark that the spatial repartition is not uniform (see Fig. 8.16).

Even if we can not prove that LPUF performances are related to its location in the FPGA, we can remark that some areas are more favorable for good results. The upper left part of the FPGA is interesting since it groups a high-performance set of LPUFs. It is also interesting to denote that the LPUFs placed near to components dedicated to the UART and the LPUF controller have bad results compared to

the others. For example, LPUFs 57, 63, 69 and 75 have a *dprime* value below the average. We can then make the hypothesis that LPUFs placed-and-routed in high activity areas can be perturbated and provide lower performances.

We can then conclude that it will be finally better to choose an isolated area to place and route the LPUF. The extremities of the FPGA seems to be a good choice.

8.5 PUFKY Based on LPUF

The previous sections in this chapter showed that LPUF is suitable to be implemented in a secure key generator. This PUFKY is made of a PUF IP with a Fuzzy Extractor, which has the role of correcting the noise at the output of the PUF, and extracting a 256-bit key. To integrate this PUFKY in a functional and autonomous *secure boot*, it is needed to implement the PUF IP with the Challenge Controller and the Lehmer-Gray component. A global finite-state machine controls all the components.

8.5.1 Experimental Results of the PUF IP

It is first important to check that the PUF has sufficient performances to be implemented.

The intra-distance graph (see Fig. 8.17) confirms that the measured data follows a normal distribution, with an average equal to 2.72 %. Because the LPUF is located far away from the other components, it is not perturbated and has good results.

Hori criterias (see [3] and Table 8.7) confirms that the data coming from the PUF has a 95 % entropy.

Moreover, the error rate remains stable. We can then deduce that the implemented error-correcting code must be able to correct 8 bits on 88, with a 100 % success rate. To summarize, this PUF has sufficient good properties to be used in a PUFKY component.

8.5.1.1 Remarks on the Challenges Choice

During the measurements of this PUF, we remarked that some challenges allowed to get more stable responses. Thanks to iterative and repeated tests (which can be easily automated), we have selected 10 optimal challenges. The reason why the PUF reacts better with some challenges is that the bit values of a challenge are in fact control bits. These values modify the oscillation occurring inside the LPUFs. There are some conditions to respect on the challenge choice to get a low noise response.

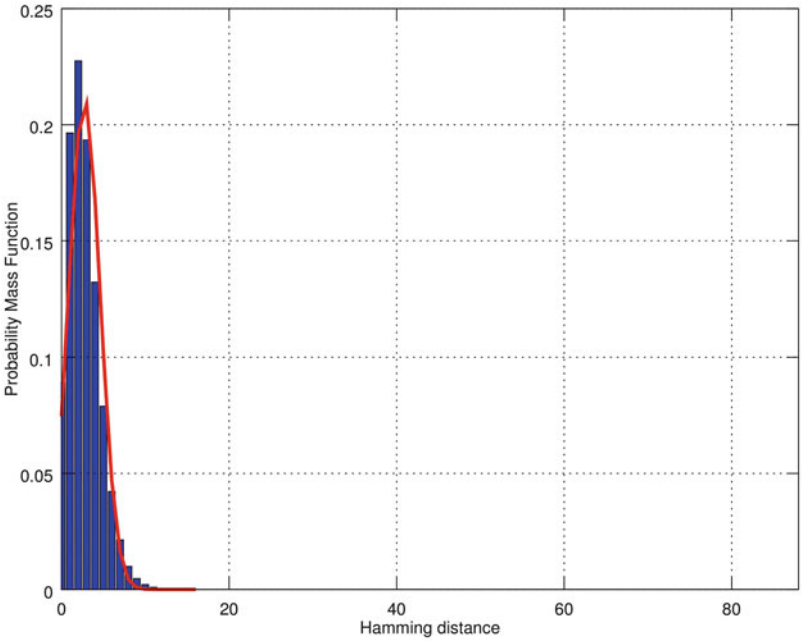


Fig. 8.17 IP PUF (100 challenges, 101 iterations)

Table 8.7 Hori criterias of the PUF IP

Criteria		Average	Standard deviation
Probability	*p*	*0.4682*	*0.4834*
Randomness	*H*	0.9511	NC
Steadiness	*S*	0.9615	0.1444
Correctness	*C*	0.9558	0.1584
Diffuseness	*D*	0.9188	0.1491

Figures 8.18 and 8.19 show that on 10 challenges, the error rate is equal to 2.41 % on average for overall the challenges and 0.6 % for 10 selected challenges. These 10 challenges have been integrated in our *secure boot*.

8.5.2 Fuzzy Extractor Characteristics

8.5.2.1 Secure Sketch

Repetition Code

A PUFKY (and consequently the error correcting code) must be lightweight and fast. A first idea that seems to be convenient at first sight is to implement a

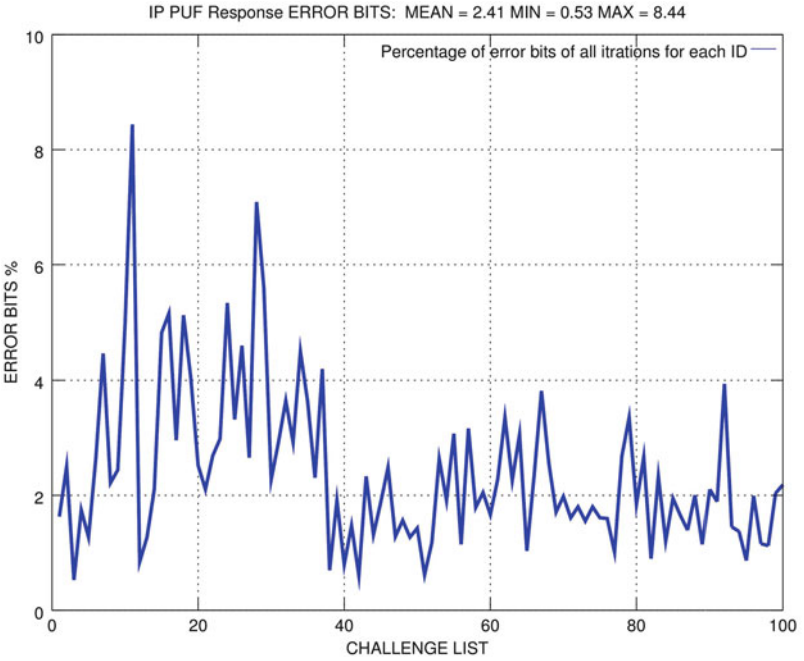


Fig. 8.18 Error percentage before challenge selection

repetition code. Thanks to a majority vote, it chooses the majority over 5 responses coming from the same challenge. With this system, the error rate decreases by half (before: 0.6 %, after: 0.37 %, see Fig. 8.20).

Hamming Code (11,15)

To correct the remaining erroneous bits, we must use an error-correcting code which uses $(n-k)$ parity bits which allows to reconstruct the initial response. We can use for this a linear correcting code which detects and correct the errors. Since we have a low error rate after the repetition code, we can use a very simple ECC like Hamming code.

Hamming codes [6] allows to detect two errors and corrects one every k bits. To do so, it uses additional information computed and stored in the generation phase of the PUFKY. These data are re-used during the reproduction phase: it is the $(n-k)$ parity bits. Thus, to optimize correction process, we have finally chosen to implement a Hamming(11,15). This code is able to detect two errors every 11 response bits and correct one thanks to the syndrome computed with the parity bits.

The Hamming ECC generates $8\times(15-11) = 32$ parity bits for a 88-bit response. We obtain a non-noisy response, usable to generate a key. Finally, for a given

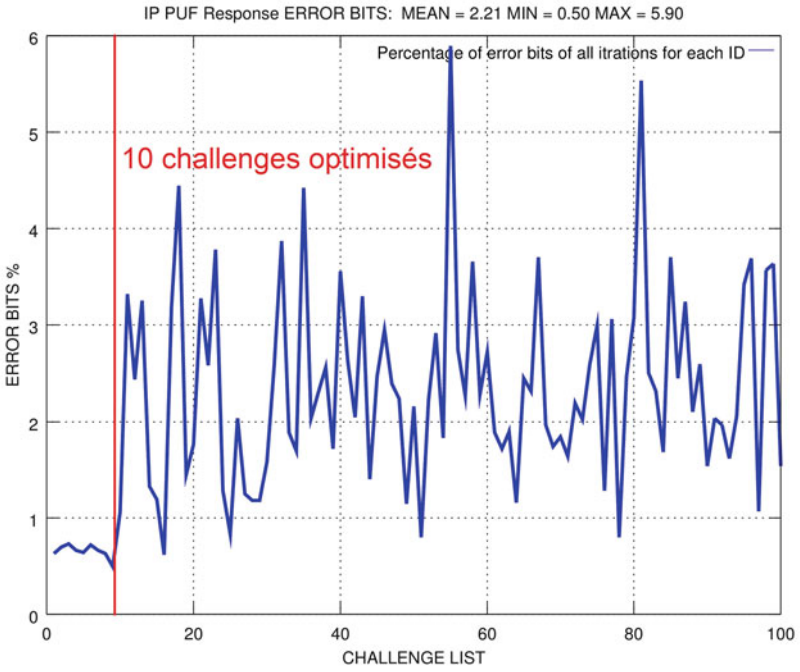


Fig. 8.19 Error percentage after challenge selection

response, there is $88 - 32 = 56$ independent bits, which means that a challenge generates 56 entropy bits. It is thus needed at least 5 challenges to generate 256 entropy bits.

Once the Hamming(11,15) code applied (see Fig. 8.21), we can see that it remains a very low average error rate (0.01 %).

To correct this last default, two solutions are possible:

- Apply a smaller Hamming code $C(4, 7)$, but the remaining entropy would be equal to 22 bits (instead of 56). We would have then to use 12 challenges to obtain 256 entropy bits. But if the number of challenges increases, the probability to obtain a noisy response would increase too. The system would then go in a vicious circle since what advantages a PUFKY part would render more vulnerable another. Using this ECC would be a drawback for the key generation.
- A second more simple solution would consist to re-iterate a defective challenge during the generation phase. Hamming code has the possibility to detect two errors on 11 bits. It could then inform the system that a measured response during the generation phase is not correct and too noisy to be corrected in a key reproduction process. The system could then measure once again our challenge which allows to correct the last errors with a 100 % success rate. That is the method we finally used in our PUFKY architecture.

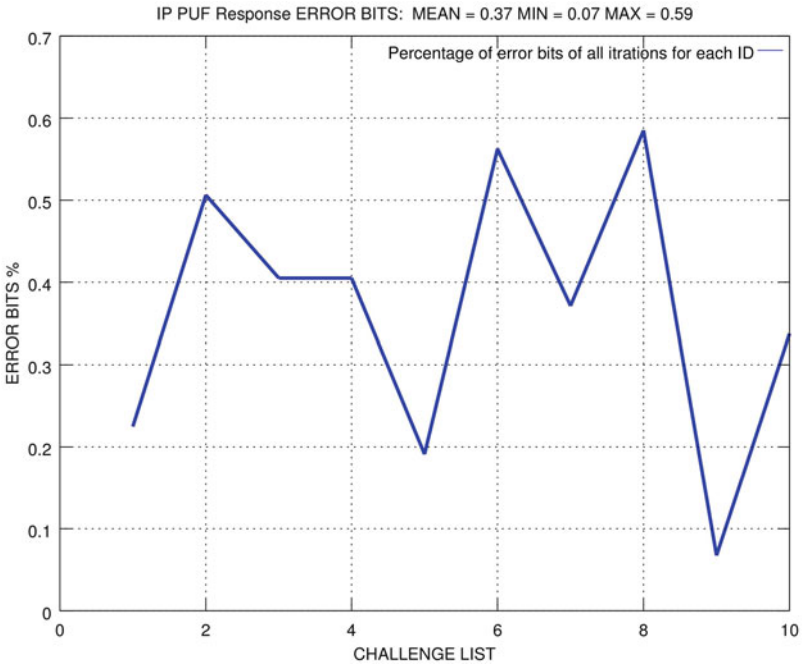


Fig. 8.20 Error percentage with a repetition code using 5 votes

8.5.2.2 The Entropy Accumulator

This part of the *Fuzzy Extractor* is implemented with a cryptographic hash function SHA-3. This hash function receives 5 corrected responses, i.e. $5 \times 88 = 440$ bits and it produces a 256 bits response which corresponds to the key generated by the PUFKY.

To guarantee the integrity of the generated key in generation and reproduction phase, we must check that there is at least 256 entropy bits in the 440 bits generated by the PUFKY:

$$[\,0.95 \times dim\,\{5 \times 88\}\,] - dim\,\{5 \times 8 \times (15-11)\} = 258 \geq 256 = dim\,\{KEY\} \tag{8.11}$$

8.5.3 Final Architecture of the PUFKY

A PUFKY based on the *LPUF* guarantees that a 256-bit key is provided to the *secure boot*. The developed general architecture of the PUFKY can be depicted in Fig. 8.22.

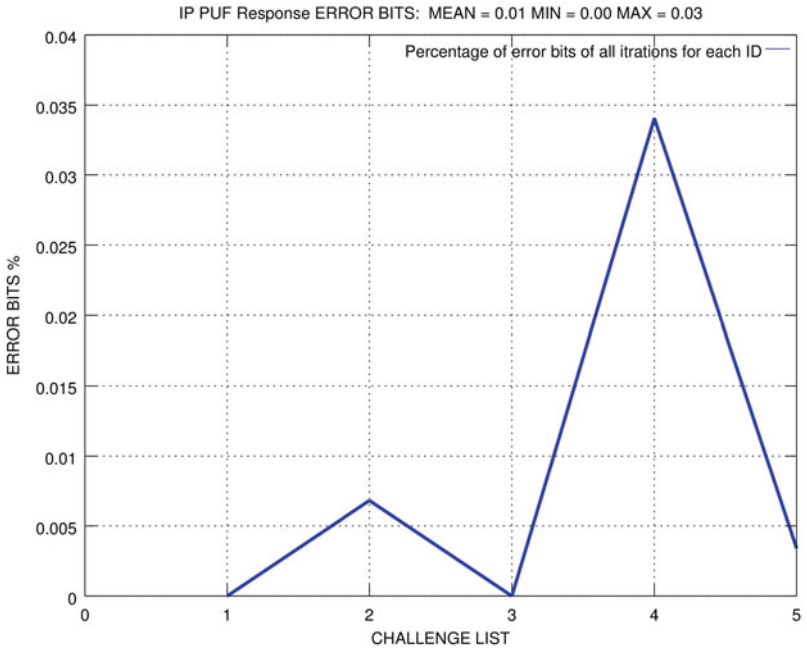


Fig. 8.21 Error percentage with a Hamming(11,15)

8.6 Summary of Our PUFKY Implementation

We have finally implemented our PUFKY IP for a *secure boot* application in a Xilinx ML507 board based on Linux (see Fig. 8.23).

This *secure boot* does not need to store a key because it is generated by the PUFKY. Moreover, this key is unique and unclonable since it comes from a PUF. Operating Systems not previously signed by our platform can not run, and no clone of the System-on-Chip can execute our Linux version previously ciphered and signed by the board.

Concerning the PUFKY part, we have successfully implemented a 100 % hardware architecture on FPGA. This PUFKY is lightweight and relatively fast for our application (it takes only 6 s on the overall boot process).

Finally, it is interesting to compare our results with the PUFKY developed by Maes et al. [7]. We can summarize this comparison in Table 8.8.

We can then remark that our architecture only uses one PUF compared to Maes et al. which uses 56 PUFs. This explains our much bigger computation time, but it is still acceptable for a *secure boot* application. A possible solution to decrease this time would consist in implementing many LPUFs in parallel to optimize the key generation process. Another approach would consist in finding the optimal number of delay chains, challenges set, FPGA location for the LPUFs, etc.

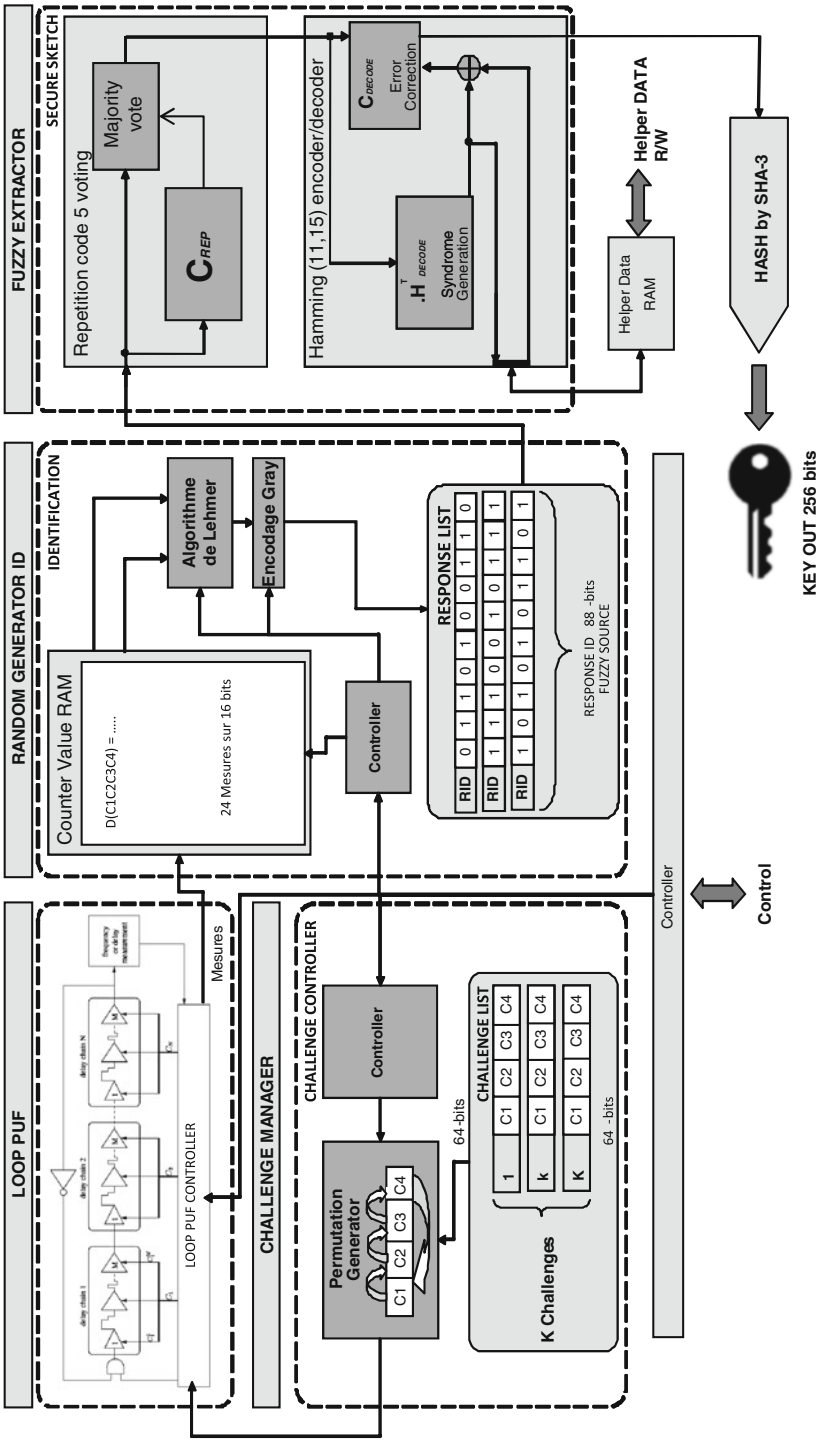


Fig. 8.22 Final architecture of the PUFKY based on a LPUF

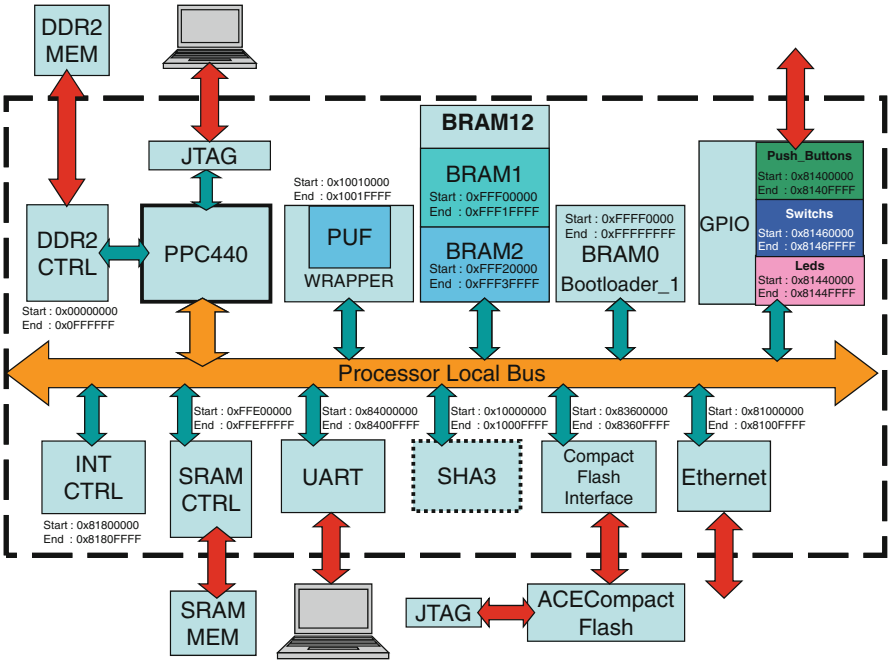


Fig. 8.23 View of the secure boot system-on-chip on ML507 board

Table 8.8 PUFKY comparaison

	Maes et al.	Francq et al.
PUF	53 ROPUFs	1 LPUF
PUF size	952 slices	24 slices
Challenges	1	5
Responses	42 bits	88 bits
PUF source	2,226 bits	440 bits
Entropy	97.95 %	95.11 %
Entropy bits	2,180 bits	418 bits
Repetition code	$C_{REP}(7,1,3)$	$C_{VOTE}(5)$
ECC	$C_{BCH}(318,174,17)$	$C_{Hamming}(11,15)$
Helper data	2,052 bits	160 bits
Remaining entropy	128 bits	258 bits
Key	128 bits	256 bits
Frequency	54 MHz	25 MHz
Computation time	5.62 ms	6 s

Conclusion

In this chapter, we have described how implementing a PUFKY in general, and using a LPUF in particular. Compared to the seminal work of Maes et al., LPUF properties allows lightweight correction of responses if the measurement time can be increased. That is the case of a secure boot application, where the key generation takes a negligible time compared to the overall boot procedure. The entropy of the key is sufficient and the size of helper data is by far smaller, which allows a more lightweight implementation.

We hope that this chapter will help designers to design quickly a PUFKY with any kind of PUF.

Acknowledgements Without the devotion and the hard work of Arnaud Lebrun under the supervision of Florentin Demetrescu (Airbus Defence and Space ─ CyberSecurity), this chapter would have never been written. We also thank Florent Lozac'h, Zouha Cherif and Jean-Luc Danger (Télécom ParisTech/Secure-IC) for providing their LPUF IP and insightful discussions.

This chapter is also dedicated to Cédric Blancher (1976–2013†), a world expert in computer security and colleague from Airbus Group Innovations.

References

1. Cherif, Z., Danger, J.L., Guilley, S., Bossuet, L.: An easy-to-design PUF based on a single oscillator: the loop PUF. In: 2012 15th Euromicro Conference on Digital System Design (DSD), Cesme, pp. 156–162. IEEE (2012)
2. Handschuh, H., Schrijen, G.J., Tuyls, P.: Hardware intrinsic security from physically unclonable functions. In: Towards Hardware-Intrinsic Security, pp. 39–53. Springer, Berlin/Heidelberg (2010)
3. Hori, Y., Yoshida, T., Katashita, T., Satoh, A.: Quantitative and statistical performance evaluation of arbiter physical unclonable functions on FPGAs. In: 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, pp. 298–303. IEEE (2010)
4. Maes, R., Van Herrewege, A., Verbauwhede, I.: PUFKY: a fully functional PUF-based cryptographic key generator. In: Cryptographic Hardware and Embedded Systems–CHES 2012, Leuven, pp. 302–319. Springer (2012)
5. Maes, R., Verbauwhede, I.: Physically unclonable functions: a study on the state of the art and future research directions. In: Towards Hardware-Intrinsic Security, pp. 3–37. Springer, Berlin/Heidelberg (2010)

6. Morelos-Zaragoza, R.H.: The Art of Error Correcting Coding. John Wiley & Sons (2006)
7. Roel, M.: Physically unclonable functions: constructions, properties and applications. Ph.D. thesis, Dissertation, University of KU Leuven (2012)
8. Ruhrmair, U., Devadas, S., Koushanfar, F.: Security Based on Physical Unclonability and Disorder. Springer, New York (2011)

Chapter 9
Fault Sensitivity Analysis at Design Time

Alessandro Barenghi, Luca Breveglieri, Andrea Palomba, and Gerardo Pelosi

Abstract Side channel attacks represent a prime threat to the security of modern digital electronic systems. Among the different attack strategies, exploiting the information leaked by the resiliency of the device to harsh working conditions has recently emerged as an exploitable mean to retrieve secret keys held in a secure device. In this chapter we will provide an introduction to the Fault Sensitivity Analysis (FSA) attack technique, together with a design time evaluation of it on two different AES S-Boxes designs. Since a critical point of performing an FSA based attack is to choose a proper model for the expected failure trend of the device, we will delineate two different approaches, one based on an a-priori modeling of it, while the other exploits an a-posteriori strategy.

9.1 Introduction

Modern embedded devices are increasingly employed to perform security critical tasks, typically involving the computation of cryptographic primitives. To this end, it is crucial to consider in the threat taxonomy, the attacks which can be lead under the assumption of having physical access to the computing device. Among those, *Side-Channel Attacks* (SCAs) represent a realistic threat to the practical security of computing systems [13, 21]. A side-channel attack exploits the information that can be retrieved either measuring physical quantity related to the ongoing computation (e.g., the power consumption of the device) or disturbing its regular functioning and observing the effects of such a disturbance. The former attack strategy yields the so-called *passive* side-channel attacks [1, 2, 8], while the latter one results in the *active* side channel attacks [4, 6, 14].

Concerning active side channel attacks, also known as *fault* attacks, the typical workflow implies inducing a controlled fault in the computation of a cryptographic primitive so to disrupt its functioning in a non catastrophic way. Such a disturbance

A. Barenghi (✉) • L. Breveglieri • A. Palomba • G. Pelosi
Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Piazza L. da Vinci, 32, 20133, Milan, Italy
e-mail: alessandro.barenghi@polimi.it; luca.breveglieri@polimi.it; palomba@elet.polimi.it; gerardo.pelosi@polimi.it

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_9

will thus affect the computation of a portion of the cipher, thus involving a limited portion of the secret parameters. Subsequently, the attacker exploits the knowledge that can be obtained either through comparing correct and faulty results of the cryptographic primitive (a technique known as Differential Fault Analysis [3, 5, 7]), or observing when such faulty results occur (also known as a *safe-error* attack).

A different exploitation of the functionality degradation of a device as a side channel is the so-called *Fault Sensitivity Analysis* [17] (FSA). FSA exploits the existing dependence between the functionality degradation of a digital device and the data being computed by it. In this chapter we will provide a description of the FSA technique, delineating its workflow. Following this, we describe a study on the feasibility of design-time prediction of the possible FSA vulnerability of a circuit, employing as a case study two different AES S-Box designs [20]. We provide insights on an alternate way to predict the functionality degradation of the device, and an experimental validation performed through post place and route logic level simulation.

9.2 Fault Sensitivity Analysis

Fault Sensitivity Analysis (FSA) was presented in [17–19] as a novel attack technique exploiting as information leakage the functionality degradation of an encrypting device. To this end, FSA employs a tunable fault induction mean, typically involving a gradual alteration of the working conditions of the digital device. For the remainder of this chapter we will consider the temporary shortening of a clock cycle to be the alteration of the working condition (sometimes denominated *clock glitch*), noting that it is possible to have gradually harsher environments through reducing more and more the clock cycle duration. Alternate alterations suitable to obtain a gradual degradation of the functionality of a digital device are: the progressive reduction of the power supply voltage level [3, 5], local laser irradiation of the die with increasing intensity [18, 19], and local EM disturbances caused through the injection of a current spike in a small coil [11].

Gradually shortening the clock period for a specific clock cycle will increase the likelihood of a failure caused by a setup-time violation of one or more logic signals, with the critical path typically failing first. We will assume that the attacker is able to shorten the clock period for the exact clock cycle when the operation he wants to gain information on is performed.

We will now detail the workflow to lead an FSA attack against a cryptographic device. The first step to perform an FSA is to observe the outputs of a device, as depicted in Fig. 9.1 for a known set of inputs $I = \{i_1, \ldots, i_n\}$. The device will compute a known cipher, employing a secret key $\mathbf{k}$ which is securely stored inside it. The attacker chooses an intermediate instruction involving known values (i.e., values which can be derived from the inputs) and a small portion of the secret key bits, denoted k from now on. From now on, we will denote the computation performed by the aforementioned instruction as $f(i_j, k)$.

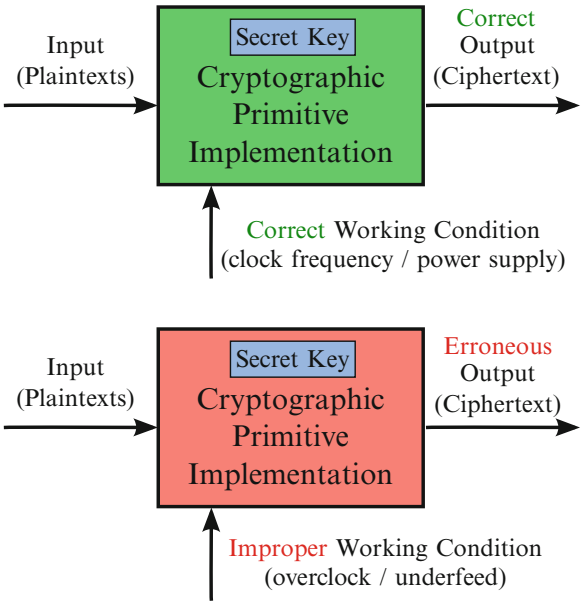


Fig. 9.1 Fault Sensitivity as a side channel: the transition point from the correct to the improper working condition reveals information on the processed data

For each one of the inputs i_j, the attacker repeatedly decreases the clock period t during the computation of $f(i_j, k_j)$, until the device is not able to output the correct result. The attacker stores the pairs $(i_j, \tilde{t})$, where it $\tilde{t}$ is the first failing clock period for the input i_j, in the set of device behavior measurements $\mathscr{M}$.

Following this step, the attacker formulates a prediction $\mathscr{P}_{\bar{k}}$ of the device behaviour for each possible value $\bar{k}$ assumed by the key bits k involved in the computation of $f(i_j, \bar{k})$. The prediction takes the form of a set of pairs (i_j, s_j), where s_j is the predicted sensitivity to faults of the computation of $f(i_j, \bar{k})$. To this end the attacker needs to devise a fault sensitivity model for the operation: such a model depends strongly on the structure of the computing circuit and should be chosen with a detailed knowledge of it. A possibility is to use the one employed in the paper proposing the FSA technique [17], i.e., the Hamming Weight of the computed value.

Once both the measurement set $\mathscr{M}$ is obtained, and all the predictions for the different values of the key portion $\mathscr{P}_{\bar{k}}$ have been computed, the attacker proceeds to correlate the predictions with the actual measurements through a statistical test of his choice. A natural choice to perform this is to compute the Pearson correlation coefficient between the measured sensitivity values and each one of the predictions, thus finding which one fits best the actual data obtained from the device. The attacker is thus able to deduce the correct value for k picking the one corresponding to the best fitting prediction. The whole secret key can be recovered piece-wise through repeating the analysis on different operations involving the remaining parts of the secret key $\mathbf{k}$.

9.2.1 Observations on FSA Feasibility

The first crucial issue to be taken into account concerning the feasibility of an FSA on a given digital device is to conceive a proper sensitivity model for the underlying digital logic. The authors of [17] note that the `and` and `or` logic gates tend to have an intrinsically data dependent fault sensitivity. This effect is due to the output signal of the gates taking a different amount of time to settle to the correct value, depending on whether one of the inputs is set to the absorbing value of the Boolean operation or not. By contrast symmetric Boolean gates, such as the `xor`, do not have an intrinsic data dependent fault sensitivity, as their settling time is always depending on the longest settling of their inputs.

In addition to the nature of the logic gates, in [16] the authors report that even small imbalances in the wires of differential logic (specifically WDDL) result in data dependencies in the fault sensitivity of the computing circuit. As a consequence, despite WDDL offers an intrinsic protection from common fault analysis, it is possible to extract information from a circuit implemented with it exploiting its sensitivity to faults. It should however be noted that it is not straightforward to infer which wires in WDDL are more sensitive to faults without being in knowledge of the post place and route details of the circuit.

In case the attacker is not able to exactly pinpoint the fault sensitive instruction, while being able to affect the working conditions of the device for a single clock cycle, it is still possible to perform correctly a FSA. In fact, it is possible for the attacker to collect fault sensitivity measurements for each clock cycle of the computation, and repeat the FSA for each set of measurements. Only one of the FSA will show a strong correlation between the actual measurements and the correct key hypothesis, while the other ones will not report viable correct candidates, due to the fault sensitivity model being wrong for any key value.

Finally, we note that the work of [23] has provided positive results on the feasibility of predicting the fault sensitivity of an hardware AES implementation at design time. In particular, the authors employed a post place and route simulation to determine the sensitivity of the implementation to FSA, employing clock period shortening as the working condition degradation strategy. The same setting was reproduced successfully on a SASEBO-R side channel evaluation board, showing a good match between the design-time predicted fault sensitivity and the actual one measured on the board.

9.3 Design Time FSA Evaluation

In this section we will provide the results of a design time evaluation of the fault sensitivity of two different AES S-box implementations, together with the results of applying a FSA to retrieve the key values from an AES implementation

employing them. To the end of performing the FSA, we will both employ the Hamming weight model suggested in [17], and devise a more efficient model for one of the two implementations.

The fault sensitivity data for the remainder of this chapter were obtained through performing post place and route logic simulation employing the circuit netlist together with the data resulting from the static timing analysis, following the same workflow employed in [23].

The first S-Box implementation is a straightforward LookUp Table (LUT) design, a typical design in high-throughput AES ASIC circuits. The optimization of the actual circuit implementing the LUT is left to the synthesis tool. The second S-Box design is based on a composite field representation, and is the one presented in [22]. The design represents the input value as two elements of $\mathbb{Z}_{2^4}$ in order to perform an efficient inversion over a smaller field, thus improving the efficiency of the implementation. After performing the field inversion, the two elements are recombined together and the affine transform is applied. The two S-Box designs have significant structural differences, which point to the possibility of the need of different fault sensitivity models.

We conducted a FSA against an implementation of the AES cipher, with a 128 bit key, considering the first SUBBYTES as the operation under attack $f(i, k)$, and taking into account one byte of the input at once, while building predictions guessing the corresponding key byte involved in the computation. In order to provide sound estimates of the correlation coefficient obtained as a result, we employed 2,000 plaintexts to perform the FSA on the targets.

Figure 9.2 reports the results of conducting the FSA on the AES implementation with a LUT-Based S-box, employing the Hamming weight of the S-Box output as the prediction of the value sensitivity. As the figure reports, the correct key hypothesis has the highest correlation coefficient (around 0.15), thus showing that the fault sensitivity model holds also for this S-Box design, despite it having a substantially different structure from the one employed by the authors of [17], who proposed the employed fault sensitivity model.

Figure 9.3 reports the results of leading the same attack against an AES implementation realized with the composite field S-Box, while still retaining the Hamming weight of the S-box output value as the prediction for the fault sensitivity of the device. As it can be seen, no clear correlation emerges from the predictions based on the Hamming weight model: the correlation for the correct key candidate (value 65) is close to zero, and the other correlation coefficients are within the range for statistical artifacts. Consequentially, we can state that the Hamming weight of the output does not provide a good fault sensitivity model for the composite field S-Box implementations.

9.3.1 An Alternative Model for S-Box Fault Sensitivity

Willing to provide a working model of the fault sensitivity of the composite field S-Box, we proceeded to analyze exhaustively its fault sensitivity behavior. To this

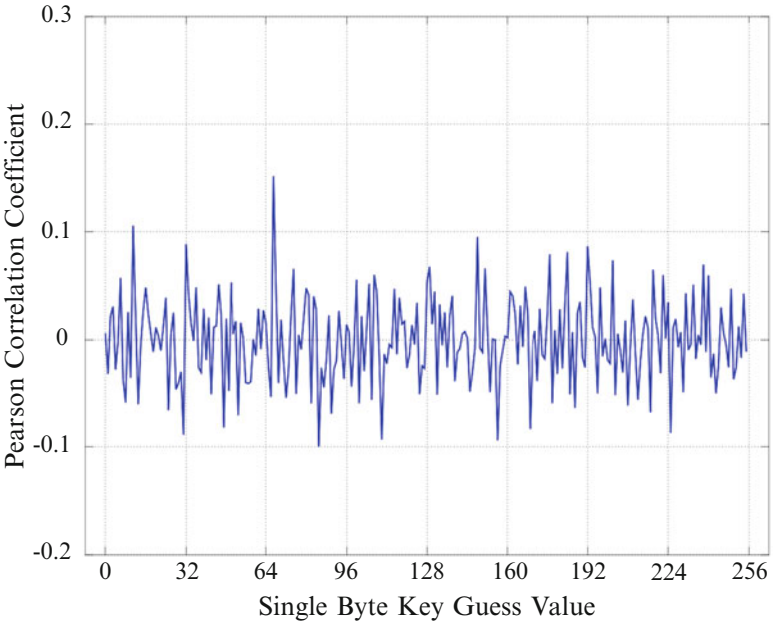


Fig. 9.2 Results of the FSA conducted against the input values of the LUT based AES S-box, employing a Hamming weight fault sensitivity model. The correct key candidate shows a significant correlation with the simulation outputs

end we simulated every possible toggling of the inputs of the S-Box circuit, i.e. we presented the S-Box with an 8-bit value, until it reached a steady state, and then toggled it to a new one for all the possible pairs of 8-bit values. The inputs and outputs of the S-Box were wired to two separate set of latches, and the clock period driving the latches was gradually shortened to measure the fault sensitivity of the S-Box transitions.

Figure 9.4, reports the results of the fault sensitivity analysis of the composite field S-Box. The measured fault sensitivities are reported in terms of the shortest clock period providing a correct output for a given steady state-input pair. The figure depicts the fault sensitivity for each pair with a marker, highlighting the points where more than one marker is superimposed. The markers are ordered depending on the Hamming weight of the input presented after the steady state of the box. The first noteworthy point of the figure is the fact that there is a significant amount of markers which point out to steady state-input pairs working at any clock frequency (the ones lying on the zero pico-seconds clock period line). These pairs are the ones that, regardless of the Hamming weight of the value, present as input the same value of the previous steady state. As no practical transition happens in these cases, the simulation deems them functioning at any clock cycle. The second noteworthy point is the fact that the transitions of the inputs to a zero value, the only one having zero Hamming weight, are significantly less sensitive to faults than all the others,

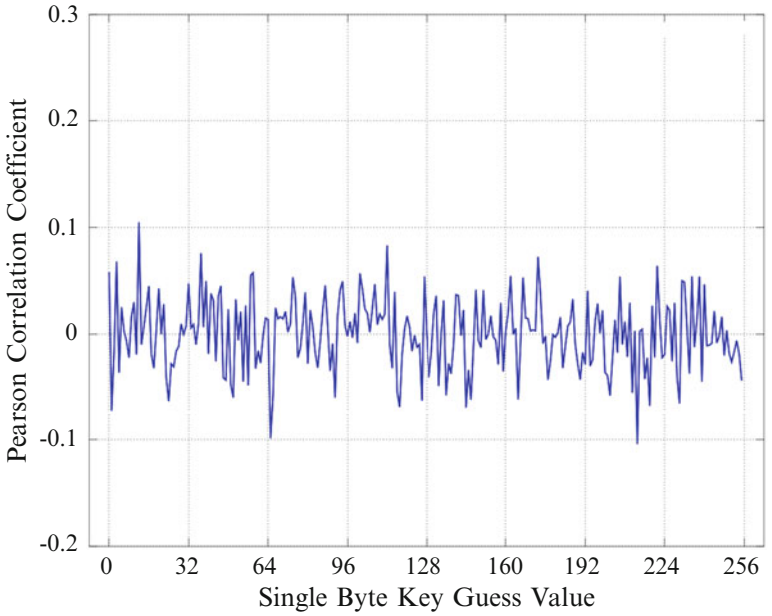


Fig. 9.3 Results of the FSA conducted against the input values of the composite field based AES S-box, employing a Hamming weight fault sensitivity model. The correct key candidate sensitivity prediction is not correlated with the outputs

regardless of the previous steady state. To fit this behavior, it is possible to devise a simple fault sensitivity prediction function which associates a zero value to the fault sensitivity when the input to the S-Box is null, and one otherwise. Such a *zero-non-zero* function is similar to the power consumption prediction function employed for composite field S-Box implementations by the authors of [12], which was devised to fit a similar peculiarity of the behavior of this S-Box design concerning its power consumption.

Figure 9.5 shows the comparison between the FSA results obtained with the zero-non-zero fault sensitivity prediction versus the ones of the Hamming weight. As it can be seen, the zero-non-zero prediction function is able to fit the fault sensitivity of the composite field S-Box well enough to allow the correct key recovery via FSA.

9.4 Template Based Fault Sensitivity Analysis

Since devising a general fault sensitivity model is challenging, due to its strong dependence on the hardware structure, we now propose a method to build them through simulation time profiling of the design behavior. The key idea is to obtain,

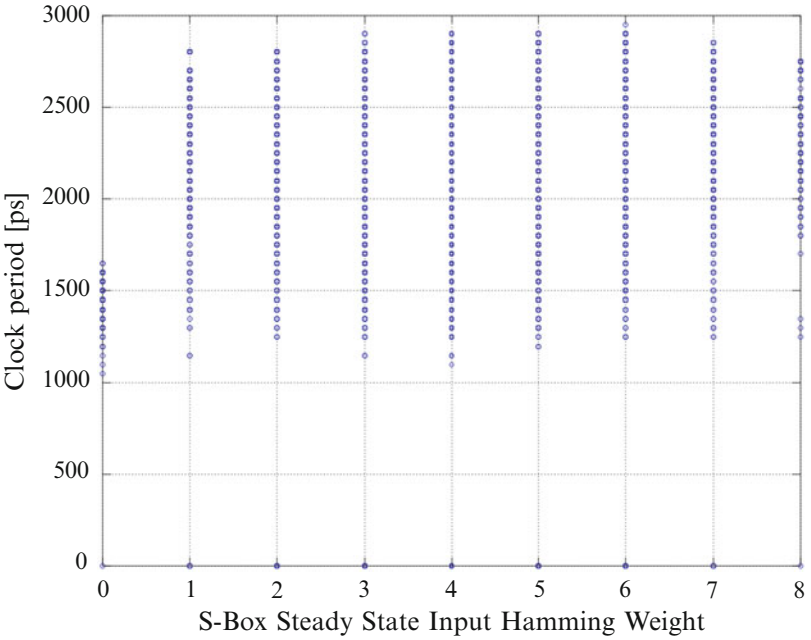


Fig. 9.4 Distribution of sensitivity levels obtained from the simulations of the composite field S-Box implementation

through profiling the behavior of the implementation, a fault sensitivity model of it [20]. This method has affinities with the so called *templates* employed in passive side channel attacks [10] in case no effective model of the power consumption or radiated EM emissions is available for the device under analysis.

In order to obtain the fault sensitivity model of the composite field S-Box through profiling it via simulation, we exploited the data reported in Fig. 9.4 to build the predictions of the fault sensitivity to a given input. Since such a model needs to depend only on the input value of the S-Box, it is necessary to choose a strategy to aggregate the different fault sensitivity values depending only on a change of the previous steady state of the cipher. To this end, three different models were built: the first considers the highest sensitivity value among all the ones obtained for a given input and a steady state, the second the lowest sensitivity, and the third the average sensitivity.

To validate the effectiveness of these choices, and discern which one is the most fruitful, we repeated the FSA on the AES circuit based on composite field S-Boxes. Figure 9.6 reports the results of such a FSA: the correlation coefficients of the correct key candidate are reported in thick lines in the figure, versus the number of inputs employed to perform the FSA. The thin lines report, for the sake of comparison, the highest coefficient among the ones obtained from the wrong key values. The depicted results show how employing either the average

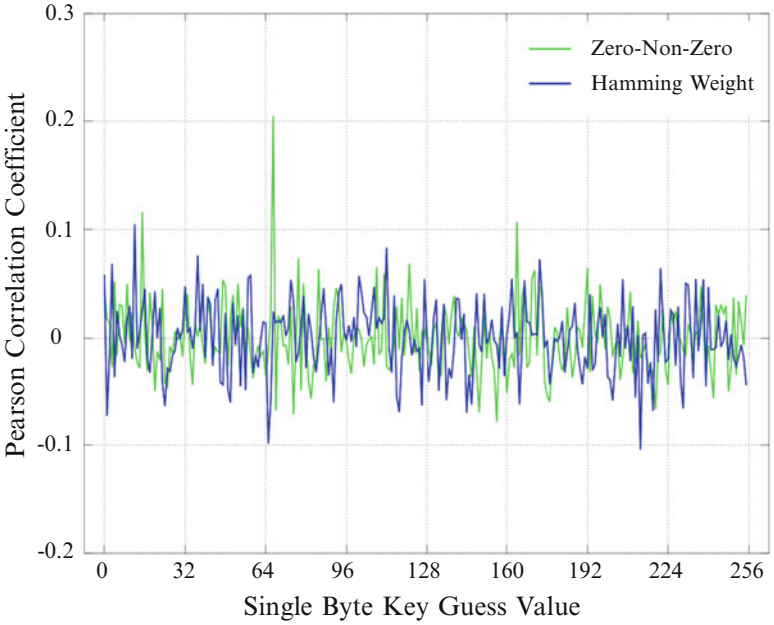


Fig. 9.5 Results of the FSA conducted against the input values of the composite field based AES S-box, comparing the zero-non-zero with the Hamming weight fault sensitivity prediction functions. The prediction for the correct key value based on the zero-non-zero fault sensitivity prediction function shows clear correlation with the simulations

or the maximum sensitivity to build the profiled model for the composite field S-Box, provide a very good match with the actual sensitivity of the whole AES implementation. In particular, the correlation coefficient for the correct key is significantly greater than the alternatives for both of them, while no significant correlation is shown with the wrong key alternatives. By contrast, employing the minimum among all the fault sensitivities of an input to the S-Box, while changing the previous steady states, as a model for the sensitivity does, is not a fruitful choice according to the results. In fact, the minimum sensitivity model shows poor correlation for the correct key alternative, and very low distinguishability from the wrong ones.

Finally, Fig. 9.7 reports the results of a direct comparison between the a-priori Hamming weight model and the one obtained keeping the average profiled fault sensitivity of the S-Box. As it can be seen, the profiled sensitivity model exceeds significantly the efficiency of the Hamming weight one in extracting information via Fault Sensitivity Analysis. In particular, it can be noticed that the correlation coefficient for the correct key candidate stabilizes around 0.31 employing it, while the highest coefficient among all the wrong key candidates does not exceed 0.18, allowing to distinguish it clearly starting from 800 inputs.

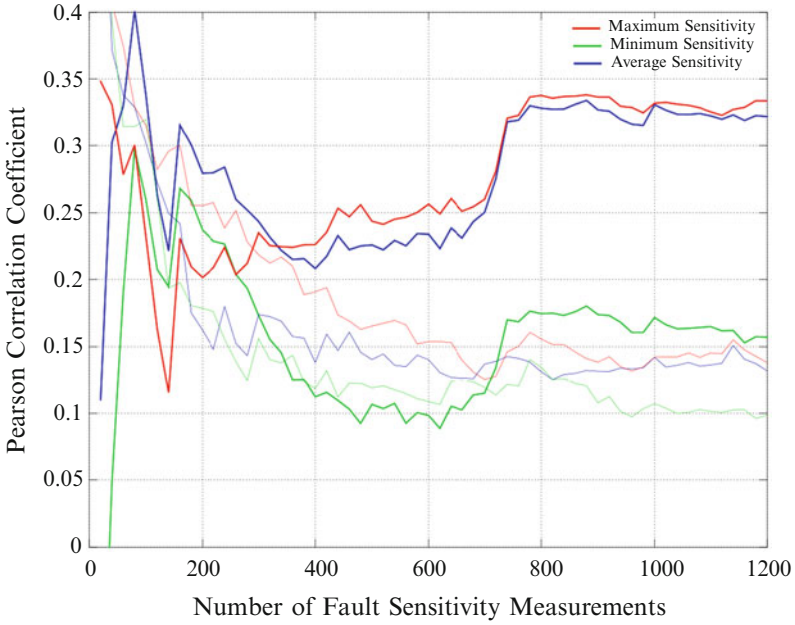


Fig. 9.6 Results of performing a FSA on an implementation of the AES cipher with composite field S-Boxes. The fault sensitivity model have been derived employing respectively the maximum (*red*), average (*blue*), and minimum (*green*) fault sensitivity level for an input byte of the S-box among all the measurements taken with all the possible different steady states. The *thick lines* represent the correlation with the correct key guess, the *thin* ones the best fitting ones among the wrong ones

Concluding Remarks

In this section we provided an introduction to the Fault Sensitivity Analysis (FSA) attack technique, together with a design time evaluation of it on two different AES S-Boxes designs. After ascertaining that the Hamming weight model for fault sensitivity is not able to provide matching predictions in the case of a composite field AES S-Box design, we proposed a new fault model, the zero-non-zero which has shown a significantly better fit to the design characteristics. Finally, we proposed a method to build a generic fault sensitivity model for a hardware design through profiling the actual device and employing the profiled data to predict the fault sensitivity of an implementation with an unknown key. We deem an interesting direction for future research to provide validation of the profiled fault sensitivity models against real hardware, possibly taking into account process variation issues, which are ever increasing with the current etching technology evolution.

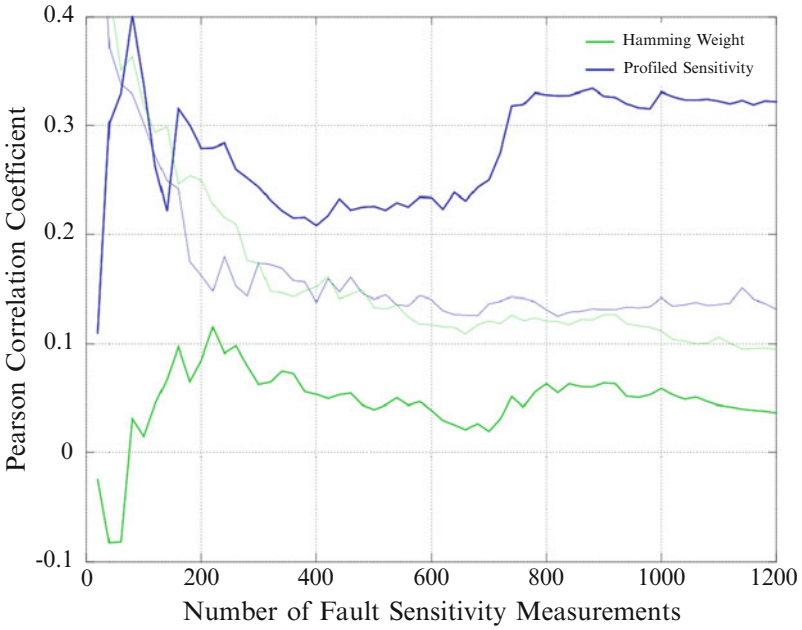


Fig. 9.7 Comparison between the FSA results obtained with a Hamming weight fault sensitivity (*green*) model with respect to the one obtained via profiling (*blue*). The *thick color lines* represent the correlation with the correct key guess, the *thin* ones the best fitting ones among the wrong ones

References

1. Agosta, G., Barenghi, A., Maggi, M., Pelosi, G.: Compiler-based side channel vulnerability analysis and optimized countermeasures application. The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1–5, 2014. ACM 2014 ISBN 978-1-4503-2730-5. In: DAC, Austin, p. 81. ACM (2013)
2. Agosta, G., Barenghi, A., Pelosi, G.: A Code morphing methodology to automate power analysis countermeasures. The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29–June 07, 2013. ACM 2013 ISBN 978-1-4503-2071-9. In: DAC, Sydney, pp. 77–82 (2012)
3. Barenghi, A., Bertoni, G., Breveglieri, L., Pellicioli, M., Pelosi, G.: Low voltage fault attacks to AES. In: Plusquellic, J., Mai, K. (eds.) HOST, Anaheim, pp. 7–12. IEEE Computer Society (2010)
4. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pellicioli, M., Pelosi, G.: Injection technologies for fault attacks on microprocessors. In: Joye and Tunstall [14], pp. 275–293
5. Barenghi, A., Bertoni, G.M., Breveglieri, L., Pelosi, G.: A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA. J. Syst. Softw. **86**(7), 1864–1878 (2013)
6. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: theory, practice, and countermeasures. Proc. IEEE **100**(11), 3056–3076 (2012)
7. Barenghi, A., Breveglieri, L., Koren, I., Pelosi, G., Regazzoni, F.: Countermeasures against fault attacks on software implemented AES: effectiveness and cost. In: WESS, Scottsdale, p. 7. ACM (2010)

8. Barenghi, A., Breveglieri, L., Santis, F.D., Melzani, F., Palomba, A., Pelosi, G.: Design time engineering of side channel resistant cipher implementations. In: Elçi, A., Pieprzyk, J., Chefranov, A.G., Orgun, M.A., Wang, H., Shankaran, R. (eds.) Theory and Practice of Cryptography Solutions for Secure Information Systems, pp. 133–157. IGI Global, Hershey (2013)
9. Bertoni, G., Gierlichs, B. (eds.): 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, 9 Sept 2012. IEEE (2012)
10. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski et al. [15], pp. 13–28
11. Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of AES. In: Bertoni and Gierlichs [9], pp. 7–15
12. Golic, J.D., Tymen, C.: Multiplicative masking and power analysis of AES. In: Kaliski et al. [15], pp. 198–212
13. Guilley, S., Meynard, O., Nassar, M., Duc, G., Hoogvorst, P., Maghrebi, H., Elaabid, A., Bhasin, S., Souissi, Y., Debande, N., Sauvage, L., Danger, J.: Vade mecum on side-channels attacks and countermeasures for the designer and the evaluator. In: 6th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2011, Athens, pp. 1–6 (2011)
14. Joye, M., Tunstall, M. (eds.): Fault Analysis in Cryptography. Information Security and Cryptography. Springer, Berlin/New York (2012)
15. Kaliski Jr., B.S.K., Çetin Kaya Koç, Paar, C. (eds.): 4th International Workshop on Cryptographic Hardware and Embedded Systems – CHES 2002, Redwood Shores, 13–15 Aug 2002. Lecture Notes in Computer Science, vol. 2523. Revised papers. Springer (2003)
16. Li, Y., Ohta, K., Sakiyama, K.: Revisit fault sensitivity analysis on WDDL-AES. In: HOST, San Diego, pp. 148–153. IEEE Computer Society (2011)
17. Li, Y., Ohta, K., Sakiyama, K.: New fault-based side-channel attack using fault sensitivity. IEEE Trans. Inf. Forensics Secur. **7**(1), 88–97 (2012)
18. Li, Y., Ohta, K., Sakiyama, K.: Toward effective countermeasures against an improved fault sensitivity analysis. IEICE Trans. **95-A**(1), 234–241 (2012)
19. Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault sensitivity analysis. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010. Lecture Notes in Computer Science, vol. 6225, pp. 320–334. Springer, Berlin/New York (2010)
20. Melzani, F., Palomba, A.: Enhancing fault sensitivity analysis through templates. In: HOST, Austin, pp. 25–28 (2013)
21. Oswald, E., Preneel, B.: A survey on passive side-channel attacks and their countermeasures for the NESSIE public-key cryptosystems. http://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/Fkulwp5-027-1.pdf
22. Rijmen, V.: Efficient Implementation of the Rijndael S-box. Technical report, Department of ESAT, Katholieke Universiteit Leuven
23. Sugawara, T., Suzuki, D., Katashita, T.: Circuit simulation for fault sensitivity analysis and its application to cryptographic LSI. In: Bertoni and Gierlichs [9], pp. 16–23

Chapter 10
Information Theoretic Comparison of Side-Channel Distinguishers: Inter-class Distance, Confusion, and Success

Annelie Heuser, Olivier Rioul, Sylvain Guilley, and Jean-Luc Danger

Abstract Different side-channel distinguishers have different efficiencies. Their fair comparison is a difficult task because many factors come into play—in particular, their intrinsic statistical properties and the quality of their estimation.

In this work, we first evaluate two related information-theoretic distinguishers: mutual information analysis and inter-class information analysis. The latter requires the same underlying probability distributions but uses a different comparing strategy. These distinguishers are not only interesting on their own and suitable for a mathematical study, but they also exhibit an example where the theoretical and empirical evaluation framework agree. The IIA was found to distinguish better than MIA in theory as well as in practice.

Moreover, we develop a new metric, called success metric, capturing the relevant parameters of the success rate, while providing more feedback about the distinguishing power. We additionally state closed-form expressions of the theoretical success metric for additive distinguisher like CPA and DPA and highlight that these expressions are much more convenient than for the theoretical success rate. In the case of a low signal-to-noise ratio (realistic practical condition), we derive the conditions on the cipher's substitution boxes (sboxes) to minimize the success metric (hence the success rate). This result supersedes a previous characterization on sboxes known as transparency order, which is derived from a metric on a distinguisher, and not from a success metric/rate. Moreover, we are also able to formulate a closed-form expression for MIA, which has not been shown before.

A. Heuser (✉) • O. Rioul • S. Guilley • J.-L. Danger
Telecom ParisTech, Institut Mines-Telecom, CNRS LTCI, Department Comelec,
46 rue Barrault, 75 634 Paris Cedex 13, France
e-mail: annelie.heuser@telecom-paristech.fr; olivier.rioul@telecom-paristech.fr;
sylvain.guilley@telecom-paristech.fr; jean-luc.danger@telecom-paristech.fr

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_10

10.1 Side-Channel Analysis

Side-channel analysis (SCA) constitutes a serious threat against modern cryptographic implementations. They exploit unintentionally emitted physical leakage—such as power consumption or electromagnetic emanation—in order to reveal secret information. The introduction of differential power analysis by Kocher et al. [16] gave rise to many developments of new attacks, countermeasures and models for the evaluation of physical security. In particular, a large variety of *distinguishers* have been proposed as statistical tests in order to discriminate the correct key. To overcome limitations such as the restriction to linear dependency between the leakage and the assumed leakage model, new types of distinguishers have been proposed.

First, mutual information analysis (MIA) was proposed by Gierlichs et al. [11]. It uses mutual information (MI) to measure the total dependency between the measurements and the leakage model. Extensive previous work [3, 17, 21, 23, 24, 38, 40] has shown that this distinguisher is indeed able to cope with non linearities between the leakage model and the measurements.

Second, to avoid explicit density estimations as required for MIA, the Kolmogorov-Smirnov (KS) test was proposed by Veyrat-Charvillon and Standaert [38] and the corresponding Kolmogorov-Smirnov analysis (KSA) was studied in [40, 42, 44]. Although it has been highlighted in [42] that KSA may have disadvantages compared to MIA, a recent study [44] has identified variants of KSA that may perform better than MIA in some circumstances.

In [18], the authors suggested an alternative *inter-class* Kolmogorov-Smirnov analysis (IKSA) that compares the conditional distributions between themselves instead of comparing them with the global distribution of the leakage. This novel approach is shown to be of a different nature (non equivalent), and can outperform KSA in terms of success rate.

Similar ideas have also emerged in the literature: The single-bit DPA [16] can already be seen as a comparison of (means of) different classes without referring to the marginal distribution. Moreover, in [2] a cluster approach has been introduced that compares the inter- and intra-class variance of conditional classes. Also, in [39] a copula-based distinguisher has been introduced that compares each conditional distribution internally without referring directly to the global leakage distribution.

It is important to note that in general, a distinguisher's performance also depends on the choice of the leakage model. As pointed out in [43] a distinguisher would fail to distinguish if the model consists of a bijective function of the secret and plaintext. Therefore, in this chapter, we restrict ourselves to leakage models for which the studied distinguishers are able to distinguish.

Because so many types of side-channel distinguishers have become available, their fair evaluation and comparison is an important topic. One cannot rely on one single experiment carried out on raw leakage measurements from one particular device to draw unequivocal conclusions about the relative efficiency of competing distinguishers (see e.g., the discussion in [33]). Therefore, we seek to compare

statistical procedures and methodologies in *ideal* scenarios with clearly defined and fixed leakage models, where in particular the signal-to-noise ratio can be varied as a parameter.

Now, there has been two distinct evaluation frameworks considered in the literature so far:

1. *A theoretical framework* proposed by Whitnall and Oswald [40] that uses the exact values of the distinguishers to evaluate the capability to recover the correct key hypothesis. One relevant metric is the so-called *relative margin*, that computes a normalized distance between the distinguisher's value for the correct key guess to that of its nearest rival.
2. *An empirical framework* proposed by Standaert et al. [34] in which the distinguishers are estimated on empirical data. The performance evaluation can be typically carried out using one of the following two metrics: the *success rate*, which estimates the probability of ranking the correct hypothesis first, and the *guessing entropy,* which estimates the average ranking of the correct hypothesis.

It should be emphasized that the theoretical framework is based on the *exact* computation of the distinguisher to evaluate its intrinsic distinguishing power—as if it was estimated on a infinite number of samples. In contrast, the empirical framework uses simulations or measurements to evaluate the ability of a distinguisher to succeed efficiently: it depends not only on the choice of the theoretical distinguisher, but also on the efficiency of its estimation. Roughly speaking, it can be said that the empirical framework encompasses the theoretical one plus the estimation algorithm. For this reason, it appears to be more practical. On the other hand, the theoretical framework is more amenable to a mathematical analysis, since it only involves the distinguisher's values. So far, no link between the theoretical and empirical outcomes of a given distinguisher has been shown in the literature.

10.1.1 Our Contributions

10.1.1.1 Interclass Distinguisher

As a first contribution we introduce a new information-theoretic metric, referred to as *inter-class information*, that compares conditional probability density functions between themselves. Before applying it to side-channel analysis, we first carry out a detailed mathematical study on the metric itself. In particular, we show that inter-class information (II) shares similar properties with mutual information (MI). Interestingly, both can be computed from the *same* probability density estimates. But we also prove that the two metrics are *not equivalent* with a precise definition of the term.

Next, we extend the inter-class information to the scenario of side-channel analysis and refer to the corresponding distinguisher as *inter-class information analysis* (IIA). We continue our mathematical investigation by proving *soundness* of

IIA. Finally, we use the above-mentioned frameworks to investigate the efficiency of both MIA and IIA. From the theoretical framework we select the *relative distinguishing margin* as the relevant metric. From the empiral framework we select the *success rate* as the relevant metric. The results from both frameworks agree: IIA is shown to outperform MIA for the theoretical *and* empirical metric.

10.1.1.2 Success Metric

Second, we introduce a new metric, called *success metric* (SM), which evaluates estimated distinguishers while providing more feedback about the efficiency. Therefore, the SM is more suitable when comparing and evaluating distinguishers than the currently state-of-the-art. In fact, SM relies on the estimation parameters of the distinguisher affecting the theoretical success rate. To be precise, the key features of the success metric are:

- Monotony with the success rate (theoretically and empirically);
- Quantification of the relationship between the distinguishing value of the correct key and its nearest rival;
- Consideration of the noise probability distribution function (e.g., its variance), number of measurements, and estimation method
- Simplicity of the closed-form expressions for additive distinguisher (e.g., DPA, CPA) compared to the success rate;
- Ability to derive a closed-form expression for MIA when estimated with histograms, which has not been shown for any other metric before.

Furthermore, we show further benefits of the closed-form expression of SM: We are able to connect the closed-form of the success metric for DPA/ CPA with properties of the sbox in case of a practical signal-to-noise ratio. Remarkably, unlike previous works [12, 22] we first not only derive bounds but achieve direct links, and second utilize a success rate/metric instead of only using properties of a distinguisher. However, our new metric, *transparency metric*, follows the same intuition as the transparency order introduced in [22], but is more reasonable and simple. Additionally, we are able to answer the question how the size of the keyspace is influencing the success metric and therefore the success rate.

10.1.2 Side-Channel Analysis Model

Calligraphic letters (e.g., $\mathcal{X}$) denote finite sets, capital letters (e.g., X) denote random variables taking values in these sets, and the corresponding lowercase letters (e.g., x) denote their realizations. We write $\mathbb{P}\{X = x\}$ or $p(x)$ for the probability that $X = x$ and $p(x|y) = \mathbb{P}\{X = x \mid Y = y\}$ for conditional probabilities.

Let k^* denote the secret cryptographic key, k any possible key hypothesis from the keyspace $\mathcal{K}$, and let T be the input or cipher text of the cryptographic algorithm.

The mapping $g : (\mathscr{T}, \mathscr{K}) \to \mathscr{I}$ maps the input or cipher text and a key hypothesis $k \in \mathscr{K}$ to an internally processed variable in some space $\mathscr{I}$ that is assumed to relate to the leakage X. Usually, $\mathscr{T}, \mathscr{K}, \mathscr{I}$ are taken as $\mathbb{F}_2^n$, where n is the number of bits (for AES $n = 8$).

Generally it is assumed that f is known to the attacker. A common consideration is $g(T, k) = \texttt{Sbox}[T \oplus k]$ where Sbox is a substitution box. The measured leakage X can then be written as

$$X = \psi(g(T, k^*)) + N, \tag{10.1}$$

where N denotes an independent additive noise.ψ is a device-specific deterministic function, which we assume to be known to the attacker in this contribution. For any key guess $k \in \mathscr{K}$ the attacker computes the *sensitive variable*

$$Y(k) = \psi(g(T, k)). \tag{10.2}$$

Without loss of generality we may assume that Y is centered and normalized, i.e., $\mathbb{E}\{Y\} = 0$ and $\mathrm{Var}\{Y\} = 1$, and that the values in $\mathscr{Y}$ are regularly spaced with step Δy. For ease of notation, we let $Y^* = Y(k^*)$ and $Y = Y(k)$.

10.2 A New Distinguisher Based on Intraclass Information

In this section, we introduce a new information-theoretic metric, referred to as *inter-class information*, that compares conditional probability density functions between themselves. Before applying it to side-channel analysis, we first carry out a detailed mathematical study on the metric itself. In particular, we show that inter-class information (II) shares similar properties with mutual information (MI). Interestingly, both can be computed from the *same* probability density estimates. But we also prove that the two metrics are *not equivalent* with a precise definition of the term.

Next, we extend the inter-class information to the scenario of side-channel analysis and refer to the corresponding distinguisher as *inter-class information analysis* (IIA). We continue our mathematical investigation by proving *soundness* of IIA. Finally, we use the above-mentioned frameworks to investigate the efficiency of both MIA and IIA.

We review some information-theoretic tools to evaluate the dependence between two random variables X and Y, and refer to [7] for more details. We focus in this section on metrics and postpone the application to side-channel analysis to Sect. 10.4. However, since for this application one random variable (X) is continuous and the other (Y) is discrete, we adopt this convention whenever it is possible.

Let $p(x)$ be the probability density function of the continuous random variable X and $p(y) = \mathbb{P}\{Y = y\}$ be the probability mass function of the discrete random

variable Y. The corresponding *expectations* are $\mathbb{E}(X) = \int_{-\infty}^{\infty} x \cdot p(x)\,\mathrm{d}x$ and $\mathbb{E}(Y) = \sum_y y \cdot p(y)$, respectively. The *variance* is defined as $\sigma_X^2 = \mathbb{E}\{(X - \mathbb{E}(X))^2\}$, and similarly for Y. Let $p(x|y) = p(x|Y = y)$ be the conditional probability distribution of X knowing that $Y = y$ and $p(x, y)$ be the joint probability distribution of X and Y. Notice that the marginal distribution $p(x)$ becomes the average over Y of the conditional distribution $p(x|y)$:

$$p(x) = \sum_y p(x, y) = \sum_y p(y)p(x|y) = \mathbb{E}(p(x|Y)). \tag{10.3}$$

10.2.1 Information Divergence

Definition 10.1 (Kullback-Leibler divergence [7]). Let $q(x)$ be another probability distribution defined over the same space as $p(x)$. The *Kullback-Leibler divergence* of q with respect to p is defined as

$$D_{\mathsf{KL}}[p \parallel q] = \int_{-\infty}^{\infty} p(x) \cdot \log \frac{p(x)}{q(x)}\,\mathrm{d}x. \tag{10.4}$$

It is well known that $D_{\mathsf{KL}}[p \parallel q] \geq 0$ and equals zero if and only if $p(x)$ and $q(x)$ coincide. The divergence is sometimes termed "distance" in the literature although it is not a distance in the mathematical sense of the word, because it is not symmetric: $D_{\mathsf{KL}}[p \parallel q] \neq D_{\mathsf{KL}}[q \parallel p]$ and the triangle inequality is not satisfied in general. To achieve symmetry, Kullback and Hajek made the following definition:

Definition 10.2 (Symmetric Kullback-Leibler divergence). The *symmetric divergence* between distributions p and q is defined as

$$\delta_{\mathsf{KL}}(p \parallel q) = \frac{D_{\mathsf{KL}}[p \parallel q] + D_{\mathsf{KL}}[q \parallel p]}{2} \tag{10.5}$$

$$= \frac{1}{2} \int_{-\infty}^{\infty} (p(x) - q(x)) \cdot \log \frac{p(x)}{q(x)}\,\mathrm{d}x. \tag{10.6}$$

10.2.2 Conditional-to-Unconditional Metric

To evaluate the dependence between X and Y, one possibility is to compute the distance between conditional probabilities $p(x|y)$ and the unconditional probability $p(x) = \mathbb{E}(p(x|Y))$ (see Fig. 10.1). Using Kullback-Leibler divergence, we obtain

$$I(X; Y) = \mathbb{E}\big\{D_{\mathsf{KL}}[p(x|Y) \parallel p(x)]\big\} \tag{10.7}$$

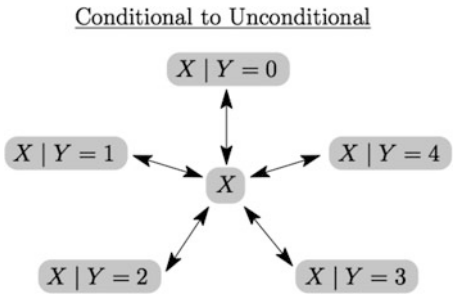


Fig. 10.1 Conditional vs Unconditional. Illustrations to compare probability distributions (the "distance" is depicted with an *arrow*)

$$= \sum_y p(y) D_{\mathsf{KL}}[p(x|y) \parallel p(x)] \tag{10.8}$$

$$= \sum_y \int_{-\infty}^{\infty} p(x, y) \cdot \log \frac{p(x|y)}{p(x)} \, \mathrm{d}x. \tag{10.9}$$

This is well-known as the *mutual information* between the two random variables X and Y. Mutual information can also be written as

$$I(X;Y) = H(X) - H(X|Y) \tag{10.10}$$

where

$$H(X) = -\int_{-\infty}^{\infty} p(x) \cdot \log p(x) \, \mathrm{d}x \tag{10.11}$$

is the (differential) *entropy* of X and

$$H(X|Y) = \sum_y p(y) \cdot H(X|Y = y) \tag{10.12}$$

$$= -\sum_y \int_{-\infty}^{\infty} p(x, y) \cdot \log p(x|y) \, \mathrm{d}x \tag{10.13}$$

is the *conditional entropy* of X knowing Y. Note that unlike the (discrete) entropy [7], differential entropy can be negative and hence should *not* be interpreted as a measure of uncertainty.[1] For more details on the relationship between differential and discrete entropy and the absolute entropy we refer to [7].

[1] Another reason is that differential entropy is not "coordinate-free" – it depends on the underlying coordinate system.

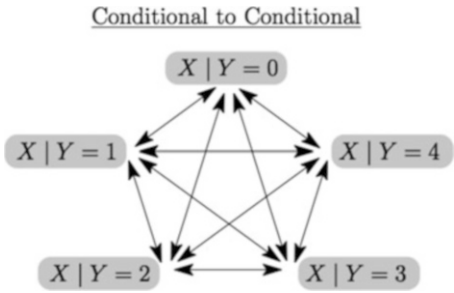


Fig. 10.2 Conditional vs Conditional. Illustrations to compare probability distributions (the "distance" is depicted with an *arrow*)

10.2.3 Conditional-to-Conditional Metric

As suggested in [18], instead of referring to the average distribution $p(x)$, a more direct strategy would be to consider *all* pairwise distances between conditional probabilities $p(x|y)$ (see Fig. 10.2). Therefore, we may replace the Kullback-Leibler divergence of $p(x|y)$ with respect to the average distribution $p(x) = \mathbb{E}(p(x|Y))$ by all Kullback-Leibler divergences between conditional probabilities $p(X|Y = y)$ and $p(X|Y = y')$ for all pairs (y, y'). This yields to the following definition.

Definition 10.3 (Inter-class information). The *inter-class information* between random variables X and Y is defined as

$$II(X;Y) = \frac{1}{2}\,\mathbb{E}\big\{D_{\mathsf{KL}}[p(x|Y=y) \parallel p(x|Y=y')]\big\} \tag{10.14}$$

$$= \frac{1}{2}\sum_{y\neq y'} p(y)p(y')D_{\mathsf{KL}}[p(x|y) \parallel p(x|y')] \tag{10.15}$$

where the summation over $y = y'$ has disappeared because divergence vanishes for identical distributions.

Proposition 10.1. *The inter-class information can also be written in terms of the symmetric Kullback-Leibler divergence as*

$$II(X;Y) = \mathbb{E}\{\delta_{\mathsf{KL}}(p(x|Y) \parallel p(x))\} \tag{10.16}$$

$$= \tfrac{1}{2}\sum_{y}\int_{-\infty}^{\infty}(p(x,y) - p(x)p(y))\log\frac{p(x,y)}{p(x)p(y)}\mathrm{d}x. \tag{10.17}$$

Proof. We show equivalence between Eqs. (10.15) and (10.16).

$$\frac{1}{2}\sum_{y\neq y'} p(y)p(y')D_{\mathsf{KL}}[p(x|y) \parallel p(x|y')]$$

$$= \frac{1}{2}\sum_{y,y'} p(y)p(y')\int p(x|y)\log\frac{p(x|y)}{p(x|y')}\,\mathrm{d}x \tag{10.18}$$

$$= \frac{1}{2}\sum_{y}\int\sum_{y'} p(y)p(y')p(x|y)\log\frac{p(x|y)}{p(x)}\,\mathrm{d}x$$

$$+ \frac{1}{2}\sum_{y'}\int\sum_{y} p(y')p(y)p(x|y)\log\frac{p(x)}{p(x|y')}\,\mathrm{d}x \tag{10.19}$$

$$= \frac{1}{2}\sum_{y} p(y)\int p(x|y)\log\frac{p(x|y)}{p(x)}\,\mathrm{d}x$$

$$+ \frac{1}{2}\sum_{y'} p(y')\int p(x)\log\frac{p(x)}{p(x|y')}\,\mathrm{d}x \tag{10.20}$$

$$= \frac{1}{2}\big(\mathbb{E}\{D_{\mathsf{KL}}[p(x|Y)\parallel p(x)]\} + \mathbb{E}\{D_{\mathsf{KL}}[p(x)\parallel p(x|Y)]\}\big) \tag{10.21}$$

$$= \mathbb{E}\big\{\delta_{\mathsf{KL}}(p(x|Y)\parallel p(x))\big\} \tag{10.22}$$

Equation (10.17) then follows easily from Definition 10.1. □

Interestingly, Eq. (10.16) is similar to Eq. (10.7) where the divergence (Definition 10.1) is replaced by the symmetric divergence (Definition 10.2). The latter is also sometimes referred to as *inter-class divergence* (see e.g., [30]).

Moreover, similarly as for mutual information, it can be expressed in terms of entropies as shown in the following proposition.

Proposition 10.2. *Let*

$$H'(X \mid Y) = -\sum_{y}\int_{-\infty}^{\infty} p(x)p(y)\cdot\log p(x|y)\,\mathrm{d}x, \tag{10.23}$$

be the conditional cross-entropy *of X knowing Y. The inter-class information can be expressed as*

$$II(X;Y) = \frac{H'(X \mid Y) - H(X|Y)}{2}. \tag{10.24}$$

Proof. We show the equivalence between Eqs. (10.17) and (10.24). Since

$$\sum_{y} p(x,y) - p(x)p(y) = 0, \tag{10.25}$$

we can remove $p(x)$ inside the logarithm in (10.17). Furthermore, since $\frac{p(x,y)}{p(y)} = p(x|y)$, we can write

$$\frac{1}{2}\sum_{x,y}(p(x,y)-p(x)p(y))\log\frac{p(x,y)}{p(x)p(y)}$$

$$=\frac{1}{2}\sum_{x,y}(p(x,y)-p(x)p(y))\log p(x|y) \tag{10.26}$$

$$=\frac{H'(X\mid Y)-H(X|Y)}{2} \tag{10.27}$$

□

It is important to notice that cross-entropy is, contrary to Eq. (10.13), averaged over the product distribution $p(x)p(y)$ instead of the joint distribution $p(x|y)p(y)=p(x,y)$.

10.3 Theoretical Analysis

Inter-class information has some important properties that are similar to well-known properties of mutual information. These are summarized in the following proposition.

Proposition 10.3. *For any two random variables X, Y:*

(a) (Symmetry) $II(X;Y)=II(Y;X)$
(b) (Independence) $II(X;Y)=0$ *if and only if X, Y are independent*
(c) (Markov Chain Inequality) *For any Markov chain $X-Y-Z$, the following hold: $II(X;Y)\geq II(X;Z)$ and $II(Y;Z)\geq II(X;Z)$*
(d) (Relation to Mutual Information)

$$2II(X;Y)=\mathbb{E}\{D_{\mathsf{KL}}[p(x|Y)\parallel p(x)]\}$$
$$+\mathbb{E}\{D_{\mathsf{KL}}[p(x)\parallel p(x|Y)]\} \tag{10.28}$$
$$=I(X;Y)+\mathbb{E}\{D_{\mathsf{KL}}[p(x)\parallel p(x|Y)]\} \tag{10.29}$$

It follows in particular that $II(X;Y)\geq\frac{1}{2}I(X;Y)$.

Proof. The symmetry is obvious from Eq. (10.17). Independency is an obvious consequence of the following well-known property of (symmetric) divergence: $D_{\mathsf{KL}}[p\parallel q]\geq 0$ and $D_{\mathsf{KL}}[p\parallel q]=0$ if and only if $p=q$ [7]. Markov Chain Inequality: Recall that $X\to Y\to Z$ forms a Markov chain if $p(z|x,y)=p(z|y)$ for all x; in other words X and Z are independent given Y [7]. Since $X\to Y\to Z$ is a Markov chain if and only if $Z-Y-X$ is a Markov chain [7], it is sufficient to prove the first inequality $II(X;Y)\geq II(X;Z)$. Furthermore we already have $I(X;Y)\geq I(X;Z)$ from the corresponding property for mutual information.

Since the latter is equivalent to the inequality $H(X|Y) \leq H(X|Z)$, thanks to Proposition 10.2, it is sufficient to prove the inequality $H'(X|Y) \geq H'(X|Z)$ for cross-entropies.

Now since $p(x|y) = p(x|y,z)$ by the Markov chain condition, it is easily checked that

$$H'(X|Y) = -\sum_{y,z} \int p(x)p(y,z) \log p(x|y,z)\,\mathrm{d}x = H'(X|Y,Z) \tag{10.30}$$

which can be rewritten as

$$H'(X|Y,Z) = \sum_{z} \int p(x)p(z) \sum_{y} p(y|z) \log \frac{1}{p(x|y,z)}\,\mathrm{d}x. \tag{10.31}$$

By the strict concavity of the logarithm, we have the following inequality

$$H'(X|Y,Z) \geq \sum_{z} \int p(x)p(z) \log \frac{1}{\sum_y p(y|z)p(x|y,z)}\,\mathrm{d}x \tag{10.32}$$

$$= \sum_{z} \int p(x)p(z) \log \frac{1}{p(x|z)}\,\mathrm{d}x = H'(X|Z) \tag{10.33}$$

Finally, the relation to mutual information is obvious from the definition. □

10.3.1 A Normal Example

In order to illustrate the difference between MI and II, we give the exact expression of $I(X;Y)$ and $II(X;Y)$ for two jointly normal random variables.[2]

Proposition 10.4. *Let the two random variables X, Y be identically distributed, zero-mean and jointly normal, with covariance matrix $\sigma^2 \left(\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right)$, where $|\rho| \leq 1$ is the correlation coefficient and σ^2 is the common variance of X and Y. One finds*

$$I(X;Y) = \frac{1}{2} \log \frac{1}{1-\rho^2} \tag{10.34}$$

$$II(X;Y) = \frac{\log e}{2} \frac{\rho^2}{1-\rho^2}. \tag{10.35}$$

Proof. Since X follows the normal density $\mathcal{N}(0, \sigma^2)$, its differential entropy is easily computed as [7]

[2]Note that, unlike in our previous definitions, the random variable Y is also continuous in this example. Thus sums have to be replaced by integrals.

$$H(X) = -\mathbb{E}\{\log p(X)\} \tag{10.36}$$

$$= \log\sqrt{2\pi\sigma^2} + (\log e)\mathbb{E}\{X^2/2\sigma^2\} \tag{10.37}$$

$$= \frac{1}{2}\log(2\pi e\sigma^2). \tag{10.38}$$

Now for every y, X given $Y = y$ follows the density $p(x|y) = \frac{p(x,y)}{p(y)}$ which is easily seen to be the normal $\mathcal{N}(\rho y, \sigma^2(1-\rho^2))$. It follows that

$$H(X|Y) = \frac{1}{2}\log(2\pi e\sigma^2(1-\rho^2)). \tag{10.39}$$

Subtracting Eq. (10.39) from Eq. (10.38) yields the announced expression for $I(X;Y) = H(X) - H(X|Y)$.

To calculate inter-class information, we use Eq. (10.24). The conditional cross-entropy can be similarly computed as

$$H'(X|Y) = -\int_{-\infty}^{\infty} p(y)\cdot\mathbb{E}\{\log p(X|y)\}\,\mathrm{d}y \tag{10.40}$$

$$= \frac{1}{2}\log(2\pi\sigma^2(1-\rho^2)) + (\log e)\int_{-\infty}^{\infty} p(y)\cdot\mathbb{E}\Big\{\frac{(X-\rho y)^2}{2\sigma^2(1-\rho^2)}\Big\}\,\mathrm{d}y. \tag{10.41}$$

Using (10.39) and expanding $\mathbb{E}\{(X-\rho y)^2\} = \mathbb{E}(X^2) + \rho^2 y^2 - 0$ inside the integral, we obtain

$$H'(X|Y) = \Big(H(X|Y) - \frac{\log e}{2}\Big) + (\log e)\frac{\sigma^2 + \rho^2\mathbb{E}\{Y^2\}}{2\sigma^2(1-\rho^2)} \tag{10.42}$$

$$= H(X|Y) + (\log e)\Big(\frac{\sigma^2 + \rho^2\sigma^2}{2\sigma^2(1-\rho^2)} - \frac{1}{2}\Big) \tag{10.43}$$

$$= H(X|Y) + (\log e)\frac{\rho^2}{1-\rho^2} \tag{10.44}$$

Subtracting $H(X|Y)$ and dividing by 2 yields the desired expression for $II(X;Y) = \frac{1}{2}(H'(X|Y) - H(X|Y))$. □

The limit case $\rho = 0$ corresponds to independent random variables X, Y in this example, while $\rho = 1$ corresponds to total dependency. From Proposition 10.4, both mutual and inter-class informations vanish when $\rho = 0$ in accordance with Proposition 10.3 (b). However, when $\rho \to 1^-$, $II(X;Y)$ is increasing to infinity much faster than $I(X;Y)$. This shows that $II(X;Y)$ is more sensitive in the dependency of the random variables. We found that this behavior is quite general for many probability distributions including the case of discrete random variables. This gives a first intuition, confirmed in the next section, why II may be more efficient than MI as a side-channel distinguisher.

10.3.2 Non-equivalence of Mutual and Inter-class Informations

Since $I(X;Y)$ and $II(X;Y)$ share similar properties (see Proposition 10.3 (a)–(c)), and since we aim to compare these two informations as side-channel distinguishers to measure dependency between the measurements and the leakage model, it is important to assert generally whether $I(X;Y)$ and $II(X;Y)$ are equivalent or not. Although this does not reflect the ability to distinguish in the context of side-channel analysis, it would give a necessary condition whether $II(X;Y)$ could be applicable. For this we need a clear definition of equivalent metrics (see e.g., [29]).

Definition 10.4 (Equivalence). Two distances $\mathscr{D}(p,q)$ and $\mathscr{D}'(p,q)$ are said to be *equivalent* if there exist finite constants $\alpha > 0$ and $\beta > 0$ such that for any p, q,

$$\mathscr{D}(p,q) \leq \alpha \cdot \mathscr{D}'(p,q) \text{ and } \mathscr{D}'(p,q) \leq \beta \cdot \mathscr{D}(p,q). \tag{10.45}$$

In particular, whenever one of two distances becomes small, so does the other and mathematically speaking, both "distances" define the same "topology".[3]

Just to illustrate the usefulness of Definition 10.4 we provide the following example.

Example 10.1. Consider the linear correlation coefficient

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \tag{10.46}$$

versus mutual information $I(X;Y)$. Although correlation implies dependence, it is possible that X and Y are linearly uncorrelated while still being dependent—take e.g., $Y = X^2$ where $X \sim \mathscr{N}(0,1)$. It follows that an inequality of the form $I(X;Y) \leq \alpha \cdot \rho(X,Y)$ *cannot* hold. Therefore, correlation and mutual information are *not* equivalent. The same conclusion goes unchanged if linear correlation is replaced by higher-order or nonlinear correlation—take e.g. $X \sim \mathscr{N}(0,1)$ and $Y = \pm X$ where the random sign is uniformly distributed and independent of X. This explains why correlation power analysis (CPA) and MIA are not equivalent.

Regarding IIA vs. MIA, Proposition 10.3 (d) shows the inequality in one direction: $I(X;Y) \leq 2 \cdot II(X;Y)$. However, we have the following.

Proposition 10.5. *Mutual information $I(X;Y)$ and inter-class information $II(X;Y)$ are* not *equivalent.*

[3]Note that this equivalence of metrics is not the same as the equivalence between distinguishers stated in [8].

Proof. It is sufficient to give the following counterexample. Consider X, Y as in Sect. 10.3.1. Letting $\lambda = \frac{1}{1-\rho^2}$ we have

$$2I(X;Y) = \log \lambda \quad \text{and} \quad 2II(X;Y) = (\lambda - 1)\log e. \tag{10.47}$$

Because the fraction $\frac{\lambda-1}{\log \lambda}$ is unbounded as $\lambda \to \infty$, letting $\rho \to 1^-$ shows that *no* inequality of the form $II(X;Y) \leq \alpha \cdot I(X;Y)$ may hold for some finite constant $\alpha > 0$. □

The fact that mutual and inter-class informations are *not* equivalent and at the same time require the estimations of the *same* conditional probability distributions $p(x|y)$ for their computation motivates for a formal comparison study in the context of side-channel analysis. This is investigated in the next section.

10.4 Side-Channel Analysis Scenario and Soundness

10.4.1 Side-Channel Scenario

There exists some necessary conditions on $Y(k)$ for MIA—and hence IIA—to be able to distinguish. In particular, [23, 43] show that there should be at least one $k \in \mathscr{K}$ such that $Y(k)$ is not an injective function of Z. Hence, if for all k, $f(\cdot, k)$ is injective the attacker has to choose φ to be non-injective. In the following, we assume that these necessary conditions are satisfied. As in [23, 24] we deduce the following scenario for wrong or correct key assumptions.

10.4.1.1 Wrong Key Assumption

The conditional distribution $p(x|y)$ of the measured leakage X knowing the predicted leakage Y is given by

$$p(x|y) = \sum_{y^*} p(y^*|y) \cdot p(x|y, y^*) \tag{10.48}$$

$$= \sum_{y^*} p(y^*|y) \cdot p(x - y^*|y) \tag{10.49}$$

$$= \sum_{y^*} p(y^*|y) \cdot p_N(x - y^*), \tag{10.50}$$

where p_N denotes the noise pdf and Eq. (10.48) follows from the law of total probability. The equivalence between Eqs. (10.49) and (10.50) follows from the fact that N is independent of the leakage predictions Y. Thus, as proved in [24], if the

key guess is incorrect we have a *nontrivial* linear mixture of shifted noise densities, whose coefficients depend on the relationship between Y and Y^*.

10.4.1.2 Correct Key Assumption

In contrast, if the key guess is correct, one obtains a Kronecker symbol $p(y^*|y) = \delta_{y,y^*}$ so that the density mixture simplifies to

$$p(x|y) = p_N(x - y^*), \tag{10.51}$$

which is simply identically distributed as $N + y^*$.

10.4.2 Soundness Proofs

Recall the following definition.

Definition 10.5 (Soundness). A given distinguisher $\mathscr{D}$ is said to be *sound* if the value of the distinguisher for the correct key k^* is strictly greater than for all other keys $k \neq k^*$:

$$\mathscr{D}(k^*) > \mathscr{D}(k) \qquad (\forall k \neq k^*) \tag{10.52}$$

Under this condition, it is an easy consequence of the law of large numbers that the corresponding success rate tends to 1 as the number of measurements increases indefinitely. For mutual information used as a side-channel distinguisher [11]: $\mathscr{D}(k) = I(X; Y(k))$, the soundness condition is expressed by the strict inequality $I(X; Y^*) > I(X; Y)$ for all $k \neq k^*$.

Proposition 10.6. *Mutual information analysis is sound for arbitrary (not necessarily Gaussian) noise.*

Proof. Moradi et al. [21] proved that $I(X; Y^*) \geq I(X; Y)$ which relies on the fact that $Y \to Y^* \to X$ forms a Markov chain [7, Thm 2.8.1]. Their paper [21] was written (as the title states) "under a Gaussian [noise] assumption" but the argument goes unchanged for non-Gaussian noise; in fact, the Markov chain condition $p(x|y, y^*) = p(x|y^*)$ relies only on the fact that N and Y are independent and not on the Gaussian nature of the noise.

To prove strict inequality, we use the fact that X given $Y = y$ is a *nontrivial* linear mixture of densities $p_N(x - y^*)$ of the same entropy as $H(N)$. Since the entropy is *strictly* concave in the probability density function [7, Thm 2.7.3][4] we have the strict inequality

[4] A well-known information-theoretic property commonly referred to as "mixing increases entropy".

$$H(X \mid Y = y) > \sum_{y^*} p(y^*|y) H(N + y^*) = H(N) \tag{10.53}$$

for all y. Taking expectations over Y yields $H(X|Y) > H(N) = H(X|Y^*)$, that is, $I(X;Y^*) > I(X;Y)$. □

For inter-class information used as a side-channel distinguisher: $\mathscr{D}(k) = II(X;Y(k))$, soundness is similarly expressed by the strict inequality $II(X;Y^*) > II(X;Y)$ for all $k \neq k^*$.

Proposition 10.7. *IIA is sound for arbitrary noise.*

Proof. Let $k \neq k^*$. By strict concavity of the logarithm (or by strict convexity of function $x \mapsto \log(1/x)$):

$$\begin{aligned}
H'(X \mid Y) = &\sum_{y,y'} p(y)p(y') \sum_{y'^*} p(y'^*|y') \\
&\times \int p_N(x - y'^*) \log \frac{1}{\sum_{y^*} p(y^*|y) p_N(x - y^*)} \, \mathrm{d}x && (10.54) \\
&< \sum_{y,y'} p(y)p(y') \sum_{y^*,y'^*} p(y'^*|y')p(y^*|y) \\
&\times \int p_N(x - y'^*) \log \frac{1}{p_N(x - y^*)} \, \mathrm{d}x && (10.55) \\
= &\sum_{y^*,y'^*} p(y'^*)p(y^*) \\
&\times \int p_N(x - y'^*) \log \frac{1}{p_N(x - y^*)} \, \mathrm{d}x && (10.56) \\
= &H'(X \mid Y^*). && (10.57)
\end{aligned}$$

Now as in the proof of Proposition 10.6, we still have $H(X|Y) > H(X|Y^*)$. Combining the two strict inequalities yields

$$\begin{aligned}
II(X;Y) &= \frac{H'(X \mid Y) - H(X|Y)}{2} && (10.58) \\
&< \frac{H'(X \mid Y^*) - H(X|Y^*)}{2} = II(X;Y^*), && (10.59)
\end{aligned}$$

which is the required soundness statement for IIA. □

10.5 Why Inter-class Information Analysis is more Discriminating than Mutual Information Analysis

In this section, we theoretically compare MIA and IIA under a Gaussian noise assumption using the scenario and the hypothesis of Sect. 10.4. We start by a theoretical investigation of $I(X;Y^*)$ and $II(X;Y^*)$, which is then extended with the help of some numerical calculation to $I(X;Y)$ and $II(X;Y)$.

10.5.1 Theoretical Comparison of $I(X;Y^*)$ and $II(X;Y^*)$

A key feature of IIA is that inter-class information is no less than mutual information for the correct key guess.[5]

Proposition 10.8. *Let X be as in Eq. (10.1) with Gaussian noise $N \sim \mathcal{N}(0,\sigma^2)$. One has*

$$II(X;Y^*) = \frac{\log e}{2} \cdot \frac{\sigma_{Y^*}^2}{\sigma^2} \tag{10.60}$$

and

$$I(X;Y^*) \leq II(X;Y^*)\,. \tag{10.61}$$

Proof. To proof Eq. (10.60) we evaluate $II(X;Y^*)$ using Eq. (10.24). Conditional cross-entropy can be written as

$$H'(X \mid Y) = \sum_y p(y) \int p(x) \log \frac{1}{p(x \mid y)}\,\mathrm{d}x. \tag{10.62}$$

Plugging the expressions $p(x) = \sum_y p(y)p(x|y)$ and $p(x|y) = \sum_{y^*} p(y^*|y)$ $p_N(x-y^*)$ yields

$$H'(X \mid Y) = \sum_{y,y'} p(y)p(y') \sum_{y'^*} p(y'^*|y')\cdot \tag{10.63}$$

$$\int p_N(x-y'^*) \log \frac{1}{\sum_{y^*} p(y^*|y)p_N(x-y^*)}\,\mathrm{d}x. \tag{10.64}$$

[5] Interestingly, it is not true that $II(X;Y) \geq I(X;Y)$ for general random variables X and Y. For example, we can find a counterexample when X, Y are binary variables with small $p(x|y)$ for all $x, y \neq 0$.

For $k = k^*$ this boils down to

$$H'(X \mid Y^*) = \sum_{y^*, y'^*} p(y^*)p(y'^*) \underbrace{\int_x p_N(x - y'^*) \log \frac{1}{p_N(x - y^*)} \, \mathrm{d}x}_{(*)} . \tag{10.65}$$

Substituting $\xi = x - y'^*$ in $(*)$ and assuming $N \sim \mathcal{N}(0, \sigma^2)$ results in

$$\begin{aligned} &\int p_N(\xi) \log \frac{1}{p_N(\xi + y'^* - y^*)} \, \mathrm{d}\xi \\ &\quad = \frac{1}{2} \log(2\pi\sigma^2) + \frac{\log(e)}{2\sigma^2} \mathbb{E}\{(N + y^* - y'^*)^2\} &(10.66)\\ &\quad = \frac{1}{2} \log(2\pi\sigma^2) + \frac{\log(e)}{2\sigma^2} \left(\sigma^2 + (y^* - y'^*)^2\right) &(10.67)\\ &\quad = H(N) + \frac{\log(e)}{2\sigma^2} (y^* - y'^*)^2. &(10.68) \end{aligned}$$

So, by letting Y'^* denote a random variable independent and identically distributed as Y^*,

$$\begin{aligned} H'(X \mid Y^*) &= H(N) + \frac{\log(e)}{2\sigma^2} \sum_{y^*, y'^*} p(y^*)p(y'^*)(y^* - y'^*)^2 &(10.69)\\ &= H(N) + \frac{\log(e)}{2\sigma^2} \mathbb{E}((Y^* - Y'^*)^2) &(10.70) \end{aligned}$$

where

$$\begin{aligned} \mathbb{E}((Y^* - Y'^*)^2) &= 2\mathbb{E}((Y^* - \mathbb{E}(Y^*))^2) &(10.71)\\ &= 2\sigma^2_{Y^*}. &(10.72) \end{aligned}$$

Combining using Eq. (10.24) and that fact that $H(X|Y^*) = H(N)$ for $k = k^*$ gives the announced formula:

$$II(X; Y^*) = \frac{H'(X \mid Y^*) - H(N)}{2} = \frac{\log e}{2} \cdot \frac{\sigma^2_{Y^*}}{\sigma^2}. \tag{10.73}$$

To prove Eq. (10.61) we use the fact that the differential entropy is maximum for normal densities [7]:

$$H(X) \leq \frac{1}{2} \log(2\pi e \sigma^2_X) \tag{10.74}$$

Since X given Y^* is normal, we obtain

$$I(X; Y^*) = H(X) - H(X|Y^*) \tag{10.75}$$

$$\leq \frac{1}{2}\log(2\pi e\sigma_X^2) - \frac{1}{2}\log(2\pi e\sigma_{X|Y^*}^2) \tag{10.76}$$

$$= \frac{1}{2}\log\frac{\sigma_X^2}{\sigma_{X|Y^*}^2} \tag{10.77}$$

$$= \frac{1}{2}\log\frac{\sigma_{Y^*}^2 + \sigma^2}{\sigma^2} \tag{10.78}$$

$$\leq \frac{\log e}{2}\frac{\sigma_{Y^*}^2}{\sigma^2} = II(X;Y^*) \tag{10.79}$$

where we have used the well-known inequality $\log x \leq (\log e)(x-1)$. □

10.5.2 Distinguishability of $I(X;Y)$ and $II(X;Y)$

We now investigate the ability to distinguish between the correct key k^* and the incorrect keys $k \neq k^*$ for MIA and for IIA. For this purpose, we use the theoretical metric given by the *relative distinguishing margin* introduced in the SCA evaluation framework in [40] and defined by

$$\texttt{RelMarg}(\mathsf{D}) = \frac{\mathsf{D}(k^*) - \max_{k\neq k^*}\mathsf{D}(k)}{\sqrt{\mathrm{Var}\,\mathsf{D}(K)}}. \tag{10.80}$$

where K is the random variable uniformly distributed in the keyspace $\mathscr{K}$.

The theoretical evaluation for both MIA and IIA involves the determination of the Gaussian density mixture of the leakage X given each possible input Z, with mean value y^* and variance σ^2. That of the conditional densities of $p(x|y)$ follow similarly for all possible values of y. Given the expressions for $p(x)$ and $p(x|y)$, we are able to compute the required entropies given in Eqs. (10.11), (10.13) and (10.23) with the help of numerical integration with arbitrary precision. To compute Eq. (10.80) we have chosen the following practical side-channel scenario:

$$Y(k) = HW(\texttt{SBox}_P^{-1}[Z \oplus k^*]) \tag{10.81}$$

$$X = Y(k^*) + N, \tag{10.82}$$

where $\texttt{SBox}_P^{-1}$ is the inverse substitution box operation in PRESENT ($\mathbb{F}_2^4 \to \mathbb{F}_2^4$), HW is the Hamming weight, and $N \sim \mathscr{N}(0,\sigma^2)$.

Figure 10.3 displays the relative distinguishing margin for various signal-to-noise ratios (SNR), defined as

$$\mathrm{SNR} = \frac{\mathrm{Var}(Y^*)}{\mathrm{Var}(N)} = \frac{2}{\sigma^2}. \tag{10.83}$$

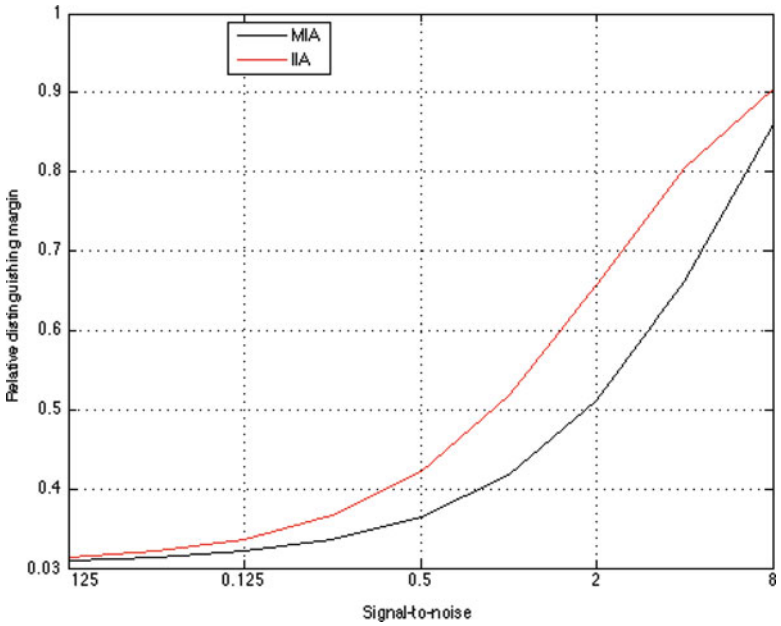


Fig. 10.3 Relative distinguishing margin for MIA (*black*) and IIA (*red*) for various SNRs

It is clearly observed that `RelMarg`(IIA) lies essentially *above* `RelMarg`(MIA) for high SNR while at smaller SNR the two curves tend to the same asymptote.

10.6 Simulation Results

In order to compare the practical and theoretical evaluations, we consider the same leakage scenario as before (Eqs. (10.81) and (10.82)). Again $N \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = \{1, 4\}$ in our simulations. Although the assumption of additive white Gaussian noise may not be always realistic, it is common in numerous works in the community.

The maximum distinguisher's value gives the key prediction $\hat{k}^*$, *viz.*,

$$\hat{k}^* = \arg\max_k I(X;Y) \text{ or } \hat{k}^* = \arg\max_k II(X;Y). \tag{10.84}$$

To compare the performance of MIA and IIA empirically we used the first-order *success rate* (SR), which we computed over a set of 230 independent experiments for $\sigma = 1$ and 120 experiments for $\sigma = 4$, where the secret key is chosen randomly for each experiment. In order to guarantee a fair comparison, we choose the same data set for both MIA and IIA.

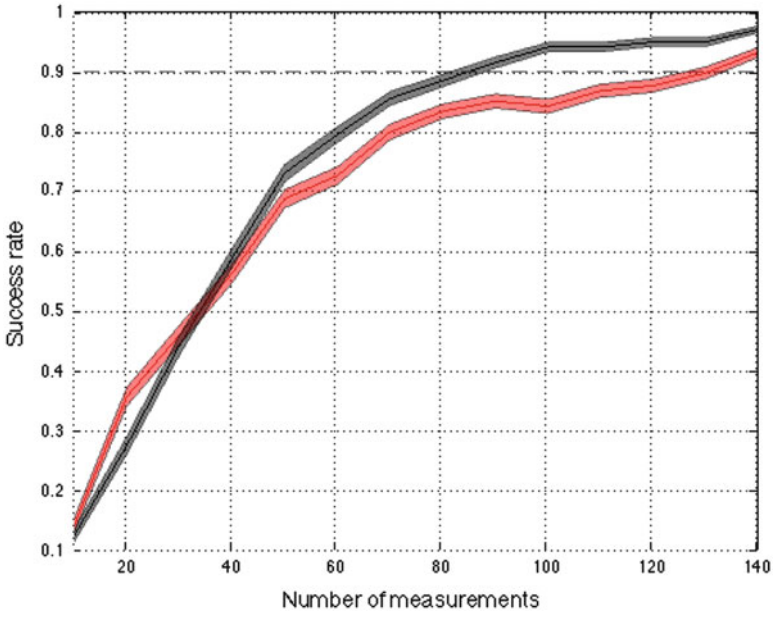


Fig. 10.4 Success rate for MIA (*red*) and IIA (*black*) with error bars using $\sigma = 1$

We used the kernel density estimation to estimate the required probability densities. The parameters were chosen as recommended in previous publications (see e.g., [3, 24, 38]). To be specific, the bandwidth was chosen according to *normal scale rule* [31] and we used the *normal* kernel.

Moreover as suggested in [18], we highlight the standard deviation of the SR by computing *error bars*. More precisely, since SR follows a binomial distribution for multiple retries R with variance $\sqrt{\frac{\mathrm{SR}(1-\mathrm{SR})}{R}}$, we obtain confidence intervals

$$\left[\mathrm{SR} - \sqrt{\frac{\mathrm{SR}(1-\mathrm{SR})}{R}}, \mathrm{SR} + \sqrt{\frac{\mathrm{SR}(1-\mathrm{SR})}{R}}\right]$$

that are drawn as error bars to provide a fair comparison.

Figure 10.4 shows the success rate with error bars for $\sigma = 1$. One can see that IIA reaches the threshold of the SR of 0.9 before MIA. The success rate for $\sigma = 4$ is displayed in Fig. 10.5, which again highlights the same classification for MIA and IIA. Interestingly, one can see that the difference between MIA and IIA is smaller for low SNR than for high SNR. Thus, the empirical results confirm our theoretical results and mathematical study made in the previous sections.

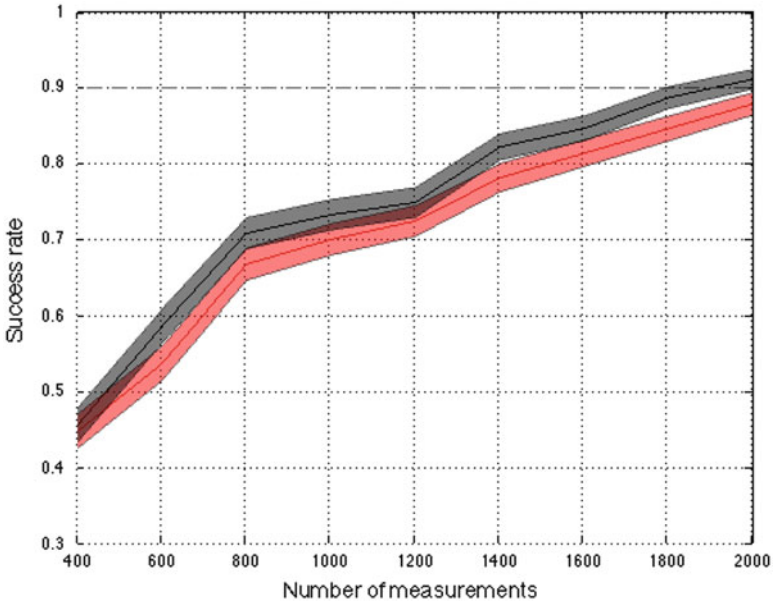


Fig. 10.5 Success rate for MIA (*red*) and IIA (*black*) with error bars using $\sigma = 4$

10.7 Comparing Side-Channel Distinguishers

10.7.1 Existing Evaluation Metrics

10.7.1.1 Comparing Empirical Distinguishers

The success rate (SR) is a classical evaluation metric when comparing empirical side-channel distinguishers $\hat{\mathscr{D}}_m(K)$. In most publications, SR is derived empirically as defined in Definition 10.6 (e.g. in [8, 18, 19]). Moreover, in [34] the authors tackled the essential question *how to compare two implementations?* or *how to compare two side-channel adversaries?* by presenting an empirical framework including the empirical success rate.

Definition 10.6 (*Empirical* success rate). Let $\hat{k} = \arg\max_k \hat{\mathscr{D}}_m(K)$ denote the key guess maximizing the experimental distinguisher $\hat{\mathscr{D}}_m(K)$ for one experiment and let $\hat{\mathbf{k}} = [\hat{k}_1, \ldots, \hat{k}_r]$ define a vector of key guesses of r independent experiments. Then the *empirical success rate* is defined as

$$\widehat{SR}(\hat{\mathscr{D}}_m) = \frac{1}{r} \sum_{i=1}^{r} \mathbb{1}_{k^* = \hat{k}_i}. \tag{10.85}$$

Even if the empirical success rate directly describes the practical outcome of a distinguisher, the given feedback is very limited. In particular, it only outputs the average probability of success without revealing influencing factors or quantifying how close the outcome of the correct key to its rivals is.

Apart from comparing the empirical SR, contributions tackled the questions on determining the *theoretical* success rate of distinguishers:

Definition 10.7 (*Theoretical* success rate). The *theoretical success rate* is defined as

$$SR(\hat{\mathscr{D}}_m) = \mathbb{P}\Big(\hat{\mathscr{D}}_m(X; Y(k^*)) > \hat{\mathscr{D}}_m(X; Y(k)) \quad (\forall k \neq k^*)\Big) \tag{10.86}$$

$$= \mathbb{P}\Big(\hat{\Delta}_m(k^*, k) > 0 \quad (\forall k \neq k^*)\Big). \tag{10.87}$$

In [27] Rivain determined the theoretical[6] SR for CPA and Bayesian attacks. Recently in [9], Fei et al. provided a *closed-form expression* for the theoretical success rate of DPA. Interestingly, their approach consists in estimating the theoretical success rate depending on the relationship between the correct and incorrect key hypothesis (named as *confusion*), the number of measurements and the SNR. Following this approach, Thillard et al. [37] extended the idea of confusion coefficients to the general case and reformulated the theoretical success rate of [27]. Thus, it is possible to determine the success rate without the need of measurements or simulations. Even more, the influencing factors of the success rate as the number of measurements, SNR and the confusion due to the leakage model are determinable. Unfortunately, the computation of the closed-form is not straightforward as mentioned in [27] and it again gives no quantification of the goodness of the distinguisher. Further, up to now only closed-forms for DPA and CPA exists.

10.7.1.2 Comparing Theoretical Distinguishers

A different approach to classify the efficiency of side-channel distinguishers has been presented in [41]. The authors aim at characterizing the behavior of theoretic distinguishers $\mathsf{D}(K)$ instead of $\hat{\mathscr{D}}_m(K)$. Thus, the distinguisher is provided with full information about the leakage distribution without the need of estimation. The framework overall consists in six metrics, however, the most common metric is the *relative distinguishing margin* (RDM) that has been used as a reference in [40,42][7]:

[6] In [27] the term exact instead of theoretical is used.

[7] Note that, in some publications, the relative distinguishing margin is also called *nearest-rival distinguishing score*.

Definition 10.8 (Relative distinguishing margin [41]). Let $\mathsf{D}(k^*)$ be the theoretical distinguishing value of the correct key and $\mathsf{D}(k)$ the theoretical distinguishing value of any incorrect key hypotheses, then the *relative distinguishing margin* RDM is defined as

$$\mathrm{RDM}(\mathsf{D}) = \frac{\mathsf{D}(k^*) - \max\limits_{k \neq k^*} \mathsf{D}(k)}{\sqrt{\mathrm{Var}(\mathsf{D}(K))}} = \min_{k \neq k^*} \frac{\mathsf{D}(k^*) - \mathsf{D}(k)}{\sqrt{\mathrm{Var}(\mathsf{D}(K))}}. \tag{10.88}$$

The RDM gives a quantified feedback about the margin between the correct key $\mathsf{D}(k^*)$ and its nearest rival, unfortunately, no link between the outcome of an empirical and a theoretical distinguisher has been shown so far. Apart from this, the denominator in Eq. (10.88) is highly dependent on the number of key hypothesis used. For example, $\sqrt{\mathrm{Var}(\mathsf{D}(K))}$ with $\mathscr{K} = \mathbb{F}_2^8$ (8-bit key hypothesis) will be smaller than for $\mathscr{K} = \mathbb{F}_2^4$ (4-bit key hypothesis) and so RDM will be smaller for smaller key spaces than vice versa, which does not seem intuitive and we prove in Sect. 10.8.2 the contrary. Thus, it is *not* possible to make reasonable comparisons between different cryptographic algorithms or implementations.

10.7.2 A Novel Approach to Compare Distinguishers

As pointed out above, both state-of-the art approaches, the SR and the RDM, have significant drawbacks, which shows the need of a new metric. Our aim is to develop a novel metric that on the one hand coincides with the empirical outcome of distinguishers, like the SR, but on the other hand gives more quantified feedback as the RDM. Our new metric, called *success metric*, captures the relevant parameters of the theoretical success rate. We provide all necessary approximations from the theoretical success rate to the success metric. In particular, we first define the failure rate as the contrary to the success rate to apply the union bound. Following, we give two different approximations identifying the same relevant influencing factors with different convergence rate and, finally, we utilize a first order approximation to achieve the success metric in Definition 10.11.

10.7.2.1 Theoretical Foundation

Complementary to the theoretical success rate (see Definition 10.86) we define:

Definition 10.9 (Failure rate). The *failure rate* is defined as

$$FR(\hat{\mathscr{D}}_m) = 1 - SR(\hat{\mathscr{D}}_m) = \mathbb{P}\big(\exists k \neq k^* \,/\, \hat{\Delta}_m(k) \leq 0\big). \tag{10.89}$$

We first use the *union bound* (Boole's inequality) to achieve an upper bound of the failure rate:

$$\mathbb{P}\big(\exists k \neq k^* \;/\; \hat{\Delta}_m(k) \leq 0\big) \leq \sum_{k \neq k^*} \mathbb{P}\big(\hat{\Delta}_m(k) \leq 0\big). \tag{10.90}$$

Next, we give two different approximations that both indicate the same properties but with different convergence rates and pre consumptions.

Definition 10.10. Let $X \sim \mathcal{N}(0,1)$. The *Q-function* is defined as

$$Q(x) = \frac{1}{2\pi} \int_x^\infty e^{-t^2/2}\, \mathrm{d}t \tag{10.91}$$

$$= \mathbb{P}(X > x). \tag{10.92}$$

Under the assumption of $\hat{\Delta}_m(k^*,k) \sim \mathcal{N}(\Delta(k^*,k), \mathrm{EV}(k^*,k))$ we use the Q-function to approximate $P\big(\hat{\Delta}_m(k^*,k) \leq 0\big)$, i.e.,

$$\mathbb{P}\big(\hat{\Delta}_m(k^*,k) \leq 0\big) \tag{10.93}$$

$$= \mathbb{P}\Big(\frac{\hat{\Delta}_m(k^*,k) - \mathbb{E}(\hat{\Delta}_m(k^*,k))}{\sqrt{\mathrm{EV}(k^*,k)}} \leq -\frac{(\Delta(k^*,k) + \mathrm{EB}(k^*,k))}{\sqrt{\mathrm{EV}(k^*,k)}}\Big) \tag{10.94}$$

$$= Q\Big(\frac{\Delta(k^*,k) + \mathrm{EB}(k^*,k)}{\sqrt{\mathrm{EV}(k^*,k)}}\Big), \tag{10.95}$$

since $Q(x) = 1 - Q(-x)$. Accordingly, if $\mathrm{EB}(k^*,k)$ is small with respect to $\Delta(k^*,k)$, we have

$$\mathbb{P}(\hat{\Delta}_m(k^*,k) \leq 0) \longrightarrow 0 \tag{10.96}$$

exponentially as

$$\frac{\Delta(k^*,k) + \mathrm{EB}(k^*,k)}{\sqrt{\mathrm{EV}(k^*,k)}} \longrightarrow \infty \tag{10.97}$$

increases for large m. We recall the Chebyshev bound [36]: Let $\rho > 0$, then

$$\mathbb{P}(X > \mathbb{E}(X) + \rho) \leq \mathbb{P}(|X - \mathbb{E}(X)| > \rho) \leq \frac{\mathrm{Var}(X)}{\rho^2}. \tag{10.98}$$

Accordingly, we achieve

$$FR = \mathbb{P}(\hat{\Delta}_m(k^*, k) \leq 0) \tag{10.99}$$

$$= \mathbb{P}\big(\hat{\Delta}_m(k^*, k) \leq \mathbb{E}\{\hat{\Delta}_m(k^*, k)\} \underbrace{-\Delta(k^*, k) - \mathrm{EB}(k^*, k)}_{-\rho}\big) \tag{10.100}$$

$$\leq \frac{\mathrm{EV}(k^*, k)}{(\mathrm{EB}(k^*, k) + \Delta(k^*, k))^2}. \tag{10.101}$$

As $\rho \longrightarrow 0$ the term $\frac{\mathrm{EV}(k^*,k)}{(\mathrm{EB}(k^*,k)+\Delta(k^*,k))^2} \longrightarrow 0$ *exponentially*.

Note that, a similar usage of the Chernov bound [6] allows to prove exponentially convergence. Further, since we achieved exponentially convergence of $\mathbb{P}(\hat{\Delta}_m(k^*, k) \leq 0)$ against 0, we use the following first order approximation

$$\sum_{k^* \neq k} \mathbb{P}(\hat{\Delta}_m(k^*, k) \leq 0) \approx \max_{k \neq k^*} \mathbb{P}(\hat{\Delta}_m(k^*, k) \leq 0). \tag{10.102}$$

Concluding, using the relationship between success and failure rate, we define the success metric as

Definition 10.11 (Success Metric (SM)).

$$\mathrm{SM}(\mathsf{D}, \hat{\mathscr{D}}_m) = \min_{k \neq k^*} \frac{\Delta(k^*, k) + \mathrm{EB}(k^*, k)}{\sqrt{\mathrm{EV}(k^*, k)}} \tag{10.103}$$

$$= \min_{k \neq k^*} \frac{\mathbb{E}\{\hat{\Delta}_m(k^*, k)\}}{\sqrt{\mathrm{Var}(\hat{\Delta}_m(k^*, k))}}. \tag{10.104}$$

Interestingly, the success metric includes the minimum distance between the correct key and its nearest-rival as the RDM, however, it is, of course, based on the estimated distinguisher and thus includes the variance of the estimated difference $\hat{\Delta}_m(k^*, k)$ in the denominator.

Remark 10.1. From Sect. 10.7.2.1 one can see that SR can be approximated from SM. More precisely,

$$\mathrm{SR} \doteq 1 - \exp\left(-\frac{1}{2}\mathrm{SM}^2\right), \tag{10.105}$$

so SM is the *first order exponent* of SR regarding the following definition of equivalence [7, page 63, Eqn. (3.76)]:

Definition 10.12. The notation $a_m \doteq b_m$ means that

$$\lim_{m\to\infty} \frac{1}{m} \log \frac{a_m}{b_m} = 0. \tag{10.106}$$

Thus, $a_m \doteq b_m$ implies that a_m and b_m are equal *to the first order* in the exponent.

As the success rate, the success metric can be derived empirically from simulations/ measurements or theoretically from closed-form expressions. In the next subsection we develop closed-form expressions for additive distinguisher (e.g., DPA, CPA). Even more, in Sect. 10.7.4 we derive a closed-form expression of the information theoretic distinguisher MIA for the success metric, which has not be done for any metric so far and cannot be straightforwardly extended to the success rate.

10.7.3 Closed-Form Expression for Additive Distinguishers

Definition 10.13 (Additive distinguisher). We call an estimated distinguisher $\hat{\mathscr{D}}_m(k)$ *additive* if it is unbiased (i.e., $\mathrm{EB}(k^*, k) = 0$) and takes the form

$$\hat{\mathscr{D}}_m(k) = \frac{1}{m} \sum_{i=1}^{m} \hat{\mathscr{D}}(X_i, Y_i(k)), \tag{10.107}$$

where $\hat{\mathscr{D}}(X_i, Y_i(k))$ is a deterministic function of the i.i.d. sequence $(X_i, Y_i(k))$ and, therefore

$$\mathbb{E}\{\hat{\mathscr{D}}_m(k)\} = \mathscr{D}(k). \tag{10.108}$$

Remark 10.2. This definition implicitly assumes that the distribution of $Y(k)$ is identical for all $k \in \mathscr{K}$. In other words, knowing the distribution of $Y(k)$ does not give any evidence about the secret (see [14, 25] for similar assumptions). Thus, $\mathrm{Var}\{Y(k)\}$ is constant for all $k \in \mathscr{K}$. Furthermore, without loss of generality we assume that the sensitive variable Y is normalized such that $\mathbb{E}\{Y(k)\} = 0$ and $\mathrm{Var}\{Y(k)\} = \mathbb{E}\{Y(k)^2\} = 1$.

Proposition 10.9. *Considering Remark 10.2 one can simplify both* $\hat{\mathscr{D}}_{m\,\mathsf{DPA}}$ *[16] and* $\hat{\mathscr{D}}_{m\,\mathsf{CPA}}$ *[4] to*

$$\frac{1}{m} \sum_{i=1}^{m} X_i Y_i(k). \tag{10.109}$$

Proof. A proof for $\hat{\mathscr{D}}_{m\,\mathsf{CPA}}$ is given in the following. As formalized in [8] $\hat{\mathscr{D}}_{m\,\mathsf{DPA}}$ and $\hat{\mathscr{D}}_{m\,\mathsf{CPA}}$ can be directly translated into each other. Recall the definition of CPA:

$$\hat{\mathscr{D}}_{m\,\mathsf{CPA}}(k) = \frac{\frac{1}{m}\sum_{i=1}^{m}(X_i - \overline{X})(Y_i(k) - \overline{Y(k)})}{\sqrt{\frac{1}{m}\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\frac{1}{m}\sum_{i=1}^{m}(Y_i(k) - \overline{Y(k)})^2}}, \tag{10.110}$$

where

$$\overline{X} = \frac{1}{m}\sum_{i=1}^{m} X_i \qquad \overline{Y(k)} = \frac{1}{m}\sum_{i=1}^{m} Y_i(k). \tag{10.111}$$

Due to Remark 10.2, (for large m) we have $\overline{Y(k)} = 0$ and $\frac{1}{m}\sum_{i=1}^{m}(Y_i(k) - \overline{Y(k)})^2 = 1$. Straightforward computation yields Proposition 10.9 for $\hat{\mathscr{D}}_{m\,\mathsf{CPA}}(k)$. For more details on CPA (and side-channel distinguisher) we refer to [32, 35]. □

To formulate a closed-form expression for the success metric for any additive distinguisher, we extend the idea of confusion similar to [37], which we call *general 2-way confusion coefficients*.

Definition 10.14 (General 2-way confusion coefficients). For $k \neq k^*$ we define

$$\kappa(k^*, k) = \mathbb{E}\left\{\left(\frac{Y(k^*) - Y(k)}{2}\right)^2\right\}, \tag{10.112}$$

$$\kappa'(k^*, k) = \mathbb{E}\left\{Y(k^*)^2\left(\frac{Y(k^*) - Y(k)}{2}\right)^2\right\}. \tag{10.113}$$

Remark 10.3. The confusion coefficient introduced in [37] is defined as $\kappa^\circ(k^*, k) = \mathbb{E}\{Y(k^*)Y(k)\}$ and we obtain the following relationship

$$\kappa^\circ(k^*, k) = 1 - 2\kappa(k^*, k). \tag{10.114}$$

Note that, our definition is consistent and a natural extension of the work in [9]. We now precise our side-channel model from Eqs. (10.1) and (10.2) in case of additive distinguishers. As these distinguishers are most usually used when the leakage X is linearly depend on Y^*, we assume $X = \alpha Y^* + N$.[8]

Proposition 10.10 (SM for CPA). *Let $\varepsilon = 2\alpha$. The success metric for any additive distinguisher takes the closed-form expression*

$$\mathrm{SM}(\mathsf{D}, \hat{\mathscr{D}}_m) = \min_{k \neq k^*} \frac{\varepsilon\kappa(k^*, k)}{\sqrt{\varepsilon^2(\kappa'(k^*, k) - \kappa^2(k^*, k)) + 4\sigma^2\kappa(k^*, k)}}\sqrt{m}. \tag{10.115}$$

Proof. We first give the following proposition.

[8] Note that, a similar model was also implicitly used in [9, 37].

Proposition 10.11. *The first two moments of* $\hat{\Delta}_m(k^*, k)$ *are given by*

$$\mathbb{E}\{\hat{\Delta}_m(k^*, k)\} = 2\alpha\kappa(k^*, k), \tag{10.116}$$

$$\mathrm{Var}(\hat{\Delta}_m(k^*, k)) = 4[\alpha^2(\kappa'(k^*, k) - \kappa^2(k^*, k)) + \sigma^2\kappa(k^*, k)]. \tag{10.117}$$

Proof. Recall

$$\hat{\Delta}_m(k^*, k) = (\alpha Y(k^*) + N)(Y(k^*) - Y(k)).$$

Since $\mathbb{E}\{Y(k^*)^2\} = 1$ (see Remark 10.2), we obtain

$$\mathbb{E}\{Y(k^*)(Y(k^*) - Y(k))\} = 1 - \mathbb{E}\{Y(k^*)Y(k)\} \tag{10.118}$$

$$= 2\mathbb{E}\{\Big(\frac{Y(k^*) - Y(k)}{2}\Big)\} \tag{10.119}$$

$$= 2\kappa(k^*, k). \tag{10.120}$$

Because N is independent of $Y(k)$,

$$\mathbb{E}\{N \cdot (Y(k^*) - Y(k))\} = \mathbb{E}\{N\} \cdot \mathbb{E}\{Y(k^*) - Y(k)\} = 0. \tag{10.121}$$

Therefore we obtain

$$\mathbb{E}\{\hat{\Delta}_m(k^*, k)\} = 2\alpha\kappa(k^*, k). \tag{10.122}$$

For the variance we obtain

$$\mathbb{E}\{\hat{\Delta}_m(k^*, k)^2\} = \mathbb{E}\{(XY^* - XY)^2\} \tag{10.123}$$

$$= 2\mathbb{E}\{N^2(Y^* - Y)^2\} + \alpha^2\mathbb{E}\{Y^{*2}(Y^{*2} - Y)^2\} \tag{10.124}$$

$$= 4\sigma^2\kappa(k^*, k) + \alpha^2 4\kappa'(k^*, k), \tag{10.125}$$

since all cross terms with N vanish. Hence, we have

$$\mathrm{Var}(\hat{\Delta}_m(k^*, k)) = \mathbb{E}\{\hat{\Delta}_m(k^*, k)^2\} - \mathbb{E}\{\hat{\Delta}_m(k^*, k)\}^2 \tag{10.126}$$

$$= 4[\alpha^2(\kappa'(k^*, k) - \kappa^2(k^*, k)) + \sigma^2\kappa(k^*, k)]. \tag{10.127}$$

□

Plugging Proposition 10.11 into the success metric given in Eq. (10.103) and considering the normalizing factor of the variance $\sqrt{m}$ (see Eq. (10.107)) directly derives Proposition 10.10. □

For DPA with one-bit variables $Y(k)$ we can further simplify the success metric such that it can be expressed directly through the SNR, number of measurements and 2-way confusion coefficient $\kappa(k^*, k)$:

Proposition 10.12 (SM for 1-bit DPA). *Let $\varepsilon = 2\alpha$, Y a one-bit variable (e.g., $Y \in \{\pm 1\}$) and $\hat{\mathscr{D}}_m(k)$ an additive distinguisher, then*

$$\mathrm{SM}(\mathsf{D}, \hat{\mathscr{D}}_m) = \frac{\sqrt{m}}{\sqrt{\max\limits_{k \neq k^*} \frac{1-\kappa(k^*,k)}{\kappa(k^*,k)} + \frac{1}{\kappa(k^*,k)\,\mathrm{SNR}}}}, \tag{10.128}$$

with $\mathrm{SNR} = \frac{\mathrm{Var}(signal)}{\mathrm{Var}(noise)} = \frac{\varepsilon^2}{\sigma^2}$, *since $\varepsilon = 2\alpha$ is the difference between X when $Y = 1$ and $Y = -1$.*

Proof. When $Y(k)$ $\forall k \in \mathscr{K}$ is a one-bit variable, we achieve the following simplification:

$$\kappa(k^*, k) = \mathbb{E}\{\big(\frac{Y(k^*) - Y(k)}{2}\big)^2\} = \mathbb{E}\{Y(k^*)^2\big(\frac{Y(k^*) - Y(k)}{2}\big)^2\} = \kappa'(k^*, k). \tag{10.129}$$

From this, Proposition 10.12 follows directly. □

Remark 10.4. Estimating the success rate from confusion coefficients includes a computation of a multivariate normal cumulative distribution function [26] for which (contrary as stated in [9]) no closed-form expression exists. Moreover, we discovered that the calculated covariance matrices[9] that directly depend on the confusion coefficients are not of full rank. This effect was similarly discovered for CPA by Rivain in [27], where the author propose to use Monte-Carlo simulation to overcome this problem.

According to Remark 10.4, we stress that the computation of the success metric as a closed-form expression is more convenient than using the closed-form expression for the success rate for DPA and CPA, since only 2-way confusion coefficients $(\kappa(k^*, k), \kappa'(k^*, k))$ without multivariate distributions are involved.

Additionally, with the help of $\kappa(k^*, k)$ we can give a closed-form expression for RDM (see Eq. (10.88)) for any additive distinguisher:

Proposition 10.13. *For additive distinguisher the* RDM(D) *can be simplified as*

$$\mathrm{RDM}(\mathsf{D}) = \frac{\min\limits_{k \neq k^*} \kappa(k^*, k)}{\sqrt{\mathrm{Var}(\kappa(k^*, K))}}. \tag{10.130}$$

[9] Namely $[\kappa(k^*, i, j)]_{(i,j) \in \mathscr{K} \setminus \{0\}}$ and $[\kappa(k^*, i) \times \kappa(k^*, j)]_{(i,j) \in \mathscr{K} \setminus \{0\}}$.

Proof Sketch: As the RDM takes as a input the theoretical value of a distinguisher D, $\kappa(k^*, k)$ directly describes the difference between $\mathsf{D}(k^*)$ and $\mathsf{D}(k)$ for any $k \in K$. Thus, Prop. 10.13 directly follows. □

The comparison of the closed-form expressions of RDM in Eq. (10.130) and SM in Eq. (10.115) again highlights the different aspects of both metrics.

10.7.4 Closed-Form Expression for Mutual Information Analysis

Definition 10.15. The Mutual Information Analysis distinguisher (MIA) [11] between a continuous variable X and a discrete variable Y is defined by

$$I(X;Y) = H(X) - H(X|Y), \tag{10.131}$$

where $H(X) = -\int_{-\infty}^{\infty} f(x) \cdot \log f(x)\,\mathrm{d}x$ is the (differential) *entropy* of X and $H(X|Y) = \sum_y p(y) \cdot H(X|Y = y) = -\sum_y p(y) \int_{-\infty}^{\infty} f(x|y) \cdot \log f(x|y)\,\mathrm{d}x$ is the *conditional entropy* of X knowing Y.

In practice, $I(X;Y)$ has to be estimated, while unlike for CPA or DPA the estimation of MIA is a nontrivial problem. For a detailed evaluation of estimation methods of mutual information distinguishers we refer to [38]. In the following, we consider the estimation with histograms in order to formulate a closed-form expression. To estimate MIA with histograms (H-MIA), one has to partition the leakage X into h distinct bins b_i of width Δx with $i = 1, \ldots, h$. Note again that, Y is already discrete.

Definition 10.16. Let $\hat{p}(x) = \frac{\#b_i}{m}$ with x falling into bin b_i and let $\hat{p}(x|y)$ be the estimated probability knowing $Y = y$, then

$$\hat{I}_m(X;Y) = -\sum_x \hat{p}(x) \log \hat{p}(x) + \sum_y \hat{p}(y) \sum_x \hat{p}(x|y) \log \hat{p}(x|y). \tag{10.132}$$

For simplification, we consider in the following only the negative conditional entropy $-\hat{H}(X|Y)$ as a distinguisher, since $\hat{H}(X)$ does not depend on a key hypothesis. Additionally, we reasonably assume that the distribution of Y is know to the attacker and thus we use $p(y)$ instead of $\hat{p}(y)$. So, H-MIA simplifies to

$$\text{H-MIA}(X,Y) = \sum_y p(y) \sum_x \hat{p}(x|y) \log \hat{p}(x|y) + \log \Delta x. \tag{10.133}$$

Note that, since we estimate the differential entropy the additional term $\log \Delta x$ arises, which is eliminated in Eq. (10.132). For more information on differential entropy and mutual information we refer to [7].

First, we develop a closed-form expression for $\mathrm{EB}\{\hat{\Delta}_m(k^*,k)\}$: Since Y is discrete the bias only arise due to the discretization of X and the limited number of measurements m. Thus, we utilize the approximations given for the bias of $\hat{H}(X)$ in [20] (3.14) to calculate $\mathbb{E}\{\hat{\mathscr{D}}_m(k)\}$ and $\mathbb{E}\{\hat{\Delta}_m(k^*,k)\}$ for H-MIA. To be specific, let h define the number of bins and Δx their width, then

$$\mathbb{E}\{\hat{\mathscr{D}}_m(k)\} = -\mathbb{E}\{\hat{H}(X|Y)\} = -\sum_y p(y)\mathbb{E}\{\hat{H}(X|Y=y)\}, \tag{10.134}$$

$$\approx -\sum_y p(y)\big[H(X|Y=y) + \frac{\Delta x^2}{24} J(X|Y=y)\big] - \frac{h-1}{2m}, \tag{10.135}$$

$$\mathbb{E}\{\hat{\Delta}_m(k^*,k)\} \approx \sum_y p(y)\big[H(X|Y=y) + \frac{\Delta x^2}{24} J(X|Y=y)\big] - \Big(\sum_{y^*} p(y^*)\big[H(X|Y^*=y^*) + \frac{\Delta x^2}{24} J(X|Y^*=y^*)\big]\Big), \tag{10.136}$$

with $J(X|Y) = \sum_y p(y) J(X|Y=y)$ and $J(X|Y=y)$ being the Fisher information $\int_{-\infty}^{\infty} \frac{[\frac{\mathrm{d}}{\mathrm{d}x} p(x|y)]^2}{p(x|y)} \mathrm{d}x$ [10].

Next, to calculate $\mathrm{Var}\{\hat{\mathscr{D}}_m(k)\}$ we use the law of total variance [15] (Eq. (10.137) $\Leftrightarrow$ Eq. (10.138)) and the approximations for the variance given in [20] (4.9) for Eq. (10.138) $\Rightarrow$ Eq. (10.139) and Eq. (10.140) $\Rightarrow$ Eq. (10.141):

$$\mathrm{Var}\{\hat{\mathscr{D}}_m(k)\} = \mathrm{Var}\{\hat{H}(X|Y)\}\} = \mathrm{Var}\{\mathbb{E}\{\hat{H}(X|Y=y)\}\} \tag{10.137}$$

$$= \mathrm{Var}\{\hat{H}(X)\} - \mathbb{E}\{\mathrm{Var}\{\hat{H}(X|Y=y)\}\} \tag{10.138}$$

$$\approx \mathrm{Var}\{H(X)\} - \frac{1}{m}\sum_y p(y)\,\mathrm{Var}\{-\log f(x|y)\} \tag{10.139}$$

$$\mathrm{Var}\{\hat{\Delta}_m(k^*,k)\} = \mathrm{Var}\{\mathbb{E}\{\hat{H}(X|Y=y\}\} - \mathrm{Var}\{\mathbb{E}\{\hat{H}(X|Y^*=y^*\}\} - 2\,\mathrm{Cov}(\mathbb{E}\{\hat{H}(X|Y=y\}\}, \mathbb{E}\{\hat{H}(X|Y^*=y^*\}\}) \tag{10.140}$$

$$\approx \frac{1}{m}\sum_y p(y)\,\mathrm{Var}\{-\log f(x|y)\} + \frac{1}{m}\sum_y p(y^*)\,\mathrm{Var}\{-\log f(x|y^*)\} - 2\,\mathrm{Cov}(\mathbb{E}\{\hat{H}(X|Y=y\}\}, \mathbb{E}\{\hat{H}(X|Y^*=y^*\}\}) \tag{10.141}$$

$$\leq \frac{1}{m}\Big(\sum_y p(y)\,\mathrm{Var}\{-\log f(x|y)\} + \sum_y p(y^*)\,\mathrm{Var}\{-\log f(x|y^*)\}\Big) \tag{10.142}$$

Using the closed-form expressions for $\mathrm{EB}\{\hat{\Delta}_m(k^*,k)\}$ and $\mathrm{EV}\{\hat{\Delta}_m(k^*,k)\}$ we formulate the following proposition.

Proposition 10.14 (SM for H-MIA).

$$\mathrm{SM}(\mathsf{D}, \hat{\mathscr{D}}_m) \lessapprox \min_{k^*\neq k} \frac{\big(\Delta(k^*,k) + \frac{\Delta x^2}{24}\big(J(X|Y) - J(X|Y^*)\big)\big)\sqrt{m}}{\sqrt{\sum_y p(y)\,\mathrm{Var}\{-\log f(x|y)\} + \sum_{y^*} p(y^*)\,\mathrm{Var}\{-\log f(x|y^*)\}}}, \tag{10.143}$$

with $\Delta(k^*,k) = H(X|Y) - H(X|Y^*)$, $J(X|Y) = \sum_y p(y) J(X|Y=y)$ *while* $J(X|Y=y)$ *is the Fisher information* $\int_{-\infty}^{\infty} \frac{[\frac{\mathrm{d}}{\mathrm{d}x} f(x|y)]^2}{f(x|y)}\,\mathrm{d}x$ *[10].*

Interestingly, the SM of MIA involves the number of traces as the $\sqrt{m}$ in the nominator like DPA and CPA, which seems reasonable.

Remark 10.5. If N is normal distributed with variance σ^2 we can further simplify $H(X|Y^*=y^*) = \frac{1}{2}\log(2\pi e\sigma^2)$ since $p(x|y^*) = p_N(x-y^*)$. Moreover, $J(X|Y^*=y) = \frac{1}{\sigma^2}$ and $\mathrm{Var}\{-\log f(x|y^*)\} = \frac{1}{2m}$.

Remark 10.6. Remarkably, the variance is approximately independent of the size of Δx. Only in extreme cases like $\Delta x = 1$ and $\Delta x \to \infty$ is affecting the variance. Also see [20] for more information. Interestingly, all linear terms have disappeared in the expression of the SM. The Eq. (10.145) is for instance empirically evaluated in [1].

10.8 Features of SM Expressions

10.8.1 Linking the Success to Properties of the Sbox

All previous studies about the relationship between the sbox properties and side-channel analysis considered the direct link between a metric on a *distinguisher* itself and the sbox. In [12], Guilley et al. use as a metric the maximal value of the distinguisher divided by its standard deviation (SNR). The authors demonstrate that for DPA the SNR is lower bounded by quantities that are expected to be large for sboxes resisting against linear differential cryptanalyses. Prouff introduces in [22],

an alternative metric for CPA, called the *transparency order*, that is defined as the difference between the maximal value of CPA and the average of all rivals. Besides, the power model is not the Hamming weight, but the Hamming distance; however, strangely enough, the sensitive variable is not the Hamming distance, but instead the average of the initial state which is exclusive-ored with all possible final states. This leakage model is, to our best knowledge, rather unusual in practice. In both previous works the relationship is only stated as an expected outcome but not proven. The results have been further investigated by Carlet in [5].

In the following, we not only bound but directly link the success metric and the sbox in case of low SNR (practical conditions). As DPA is a special case of CPA, we further concentrate on the closed-form expression of CPA and simplify Eq. (10.115) when $\sigma \gg \alpha$. More precisely,

$$\mathrm{SM}(\mathsf{D}, \hat{\mathscr{D}}_m) \approx \min_{k \neq k^*} \sqrt{\frac{4\alpha^2\kappa^2(k^*,k)m}{\sigma^2 4\kappa(k^*,k)}} \tag{10.144}$$

$$= \sqrt{\mathrm{SNR}}\sqrt{m} \min_{k \neq k^*} \sqrt{\kappa(k^*,k)}. \tag{10.145}$$

From Eq. (10.112), $\kappa(k^*,k^*) = 0$ and $\kappa(k^*,k) \geq 0$, thus the argument of the square root in Eq. (10.145) is always positive. Besides, by the Cauchy-Schwarz theorem, we also have that $\kappa(k^*,k) \leq 1$. Now, the objective to minimizing $\min_{k\neq k^*} \sqrt{\kappa(k^*,k)}$ (i.e., making side-channel attacks as hard as possible) is tantamount to maximizing $\max_{k\neq k^*} \mathbb{E}(Y(k^*)Y(k))$. In the following, we assume that Y^* and Y explicitly depend on an sbox (or inverse sbox) and a Hamming weight (w_H) leakage model[10] as for example $w_H(\mathtt{Sbox}[T \oplus k])$, so $Y(k) = \frac{1}{\sqrt{n}}\sum_{i=1}^{n}(-1)^{S_i(T\oplus k)} = \frac{1}{\sqrt{n}}(2w_H(S(T\oplus k)) - n)$ and

$$\mathbb{E}\{Y(k^*)Y(k)\} = \frac{1}{n}\sum_{i,j=0}^{n}\frac{1}{2^n}\sum_{t\in\mathbb{F}_2^n}(-1)^{S_i(t\oplus k^*)\oplus S_j(t\oplus k)}. \tag{10.146}$$

As $\forall a \in \{0,1\}, (-1)^a = 1 - 2a$, the goal to make CPA difficult is to minimize the following quantity, that we call the *transparency metric*

$$\min_{k\neq k^*} \sum_{i,j=0}^{n}\sum_{t\in\mathbb{F}_2^n} S_i(t\oplus k^*)\oplus S_j(t\oplus k). \tag{10.147}$$

Remark 10.7. Note that, for single-bit attacks ($n = 1$), the criteria of Eq. (10.147) simplifies to the *one-sided* criteria discovered in [13].

[10] One can easily extend the calculation also for the Hamming distance model.

So, minimizing the objective on the sbox in Eq. (10.147) is equivalent to minimizing $\min_{k \neq k^*} \kappa(k^*, k)$, which can be understood intuitively on the illustration of Fig. 10.6. The key corresponding to the nearest rival, i.e., $\operatorname{argmin}_{k \neq k^*} \kappa(k^*, k)$, shall have a confusion coefficient as high as possible.

To further illustrate the transparency metric and show the relationship to the *transparency order* [22], we use the same three sboxes as in [13]: Let $\oplus$ and $\odot$ be respectively the inner addition and multiplication of the Galois field $\mathbb{F}_{2^8}$ of 256 elements, then the sboxes are given by

1. A "bad" $\texttt{Sbox}[\cdot]$, termed S_1, of equation $y \mapsto a \odot y \oplus b$,
2. An "average" $\texttt{Sbox}[\cdot]$, termed S_{101}, of equation $y \mapsto a \odot y^{101} \oplus b$,
3. A "good" $\texttt{Sbox}[\cdot]$, termed S_{254} and used in AES, of equation $y \mapsto a \odot y^{254} \oplus b$.

Figure 10.7 displays the confusion coefficient for S_1, S_{101} and S_{254}. One can see, that the minimal $\min_{k \neq k^*} \kappa(k^*, k)$ is achieved by S_1, which is the hardest to attack with

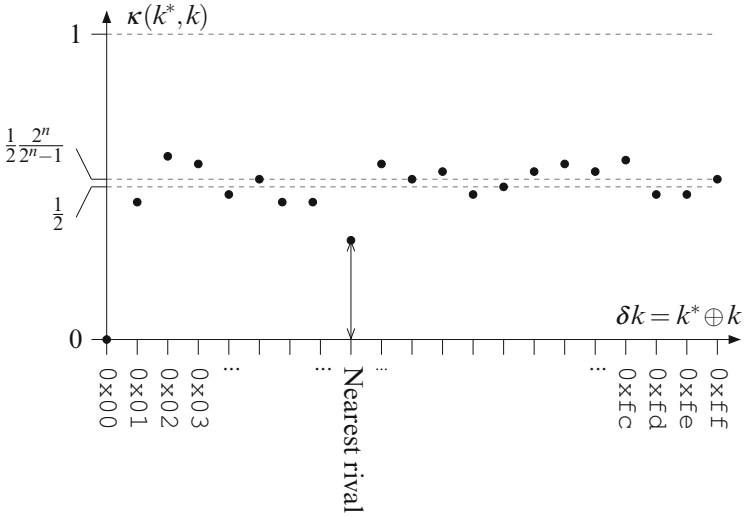


Fig. 10.6 Illustration of the confusion coefficients for CPA

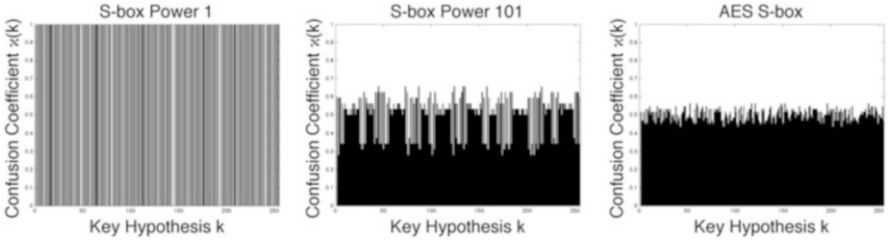


Fig. 10.7 Confusion coefficients for S_1, S_{101} and S_{254} (*courtesy of [13]*)

Table 10.1 Comparison of side-channel metrics for sboxes

	Transparency order [22]	Transparency metric (Eq. (10.147))
S_1	5.84	7,424
S_{101}	7.50	7,936
S_{254}	7.86	8,000

CPA, whereas S_{254} has the highest $\min_{k \neq k^*} \kappa(k^*, k)$ being the most vulnerable. Table 10.1 displays the transparency metric and order. The transparency metric is different from the transparency order, nonetheless, it remains consistent with it, meaning that the order of S_1, S_{101} and S_{254} is the same for both metrics and consistent with the rating through $\kappa(k^*, k)$.

10.8.2 How Does the Size of the Key Space Influence the SM/SR?

Hardware devices are known to leak approximately in Hamming distance. This makes leakage models complicated, because they involve two consecutive states of the cipher. Let us consider the example of an AES-128 computed one round per clock period. The plaintext is P, the cipher C, and the first (resp. last) round key K^1 (resp. K^{11}).

On the one hand, the uncentered and non-normalized leakage model at the first round for the byte at position 0 is:

$$Y^1(T, K^1) = w_H(T_0 \oplus \texttt{02} \cdot S(T_0 \oplus K_0^1) \oplus \texttt{01} \cdot S(T_5 \oplus K_5^1) \tag{10.148}$$

$$\oplus\, \texttt{01} \cdot S(T_{10} \oplus K_{10}^1) \oplus \texttt{03} \cdot S(T_{15} \oplus K_{15}^1)) \ , \tag{10.149}$$

where `01`, `02` and `03` are the MixColumns constants, and S is the SubBytes operation. Clearly, a guess for this model requires an hypothesis on 4 bytes of the key K^1.

On the other hand, the uncentered and non-normalized leakage model at the last round for the byte at position 0 is:

$$Y^{10}(C, K^{10}) = w_H\left(C_0 \oplus S^{-1}(C_0 \oplus K_0^{10})\right) \ , \tag{10.150}$$

where S^{-1} is the InvSubBytes operation. So, a guess for the model requires simply one hypothesis on a key byte (namely K_0^{10}). This is due to the absence of MixColumns at the last round.

The transparency order (resp. metric) of InvSubBytes is 7.85 (resp. 7,964), meaning that it is very close to that of SubBytes. So, the confusion coefficient associated to Y^1 and to Y^{10} have similar distributions, meaning that the data complexity (the number of traces m) of the attack is similar at either end of the

AES. Specifically, the minimal nonzero confusion coefficient for Y^1 is 0.468750, whereas it is 0.404297 for Y^{10}. The most crucial difference is the computational complexity, owing to the largest key space to explore at the first round.

Acknowledgements Annelie Heuser is partly founded by the Google Doctoral European Fellowship in the field of privacy.

References

1. Bhasin, S., Danger, J.-L., Guilley, S., Najm, Z.: Side-channel Leakage and Trace Compression Using Normalized Inter-class Variance, ACM, Minneapolis, Minnesota Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy, Minneapolis, Minnesota, pp. 7:1–7:9 (2014) doi: 10.1145/2611765.2611772, http://doi.acm.org/10.1145/2611765.2611772
2. Batina, L., Gierlichs, B., Lemke-Rust, K.: Differential cluster analysis. In: Clavier, C., Gaj, K. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2009, Lausanne. Lecture Notes in Computer Science, vol. 5747, pp. 112–127. Springer (2009)
3. Batina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F.X., Veyrat-Charvillon, N.: Mutual information analysis: a comprehensive study. J. Cryptol. **24**(2), 269–291 (2011)
4. Brier, É., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Cryptographic Hardware and Embedded Systems – CHES 2004, Cambridge. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004)
5. Carlet, C.: On highly nonlinear S-boxes and their inability to thwart DPA attacks. In: INDOCRYPT, Bangalore. Lecture Notes in Computer Science, vol. 3797, pp. 49–62. Springer (2005)
6. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Stat. **23**, 493–507 (1952)
7. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley-Interscience, Hoboken (2006). ISBN-10: ISBN-10: 0471241954, ISBN-13: 978-0471241959
8. Doget, J., Prouff, E., Rivain, M., Standaert, F.X.: Univariate side channel attacks and leakage modeling. J. Cryptogr. Eng. **1**(2), 123–144 (2011)
9. Fei, Y., Luo, Q., Ding, A.A.: A statistical model for DPA with novel algorithmic confusion analysis. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2012, Leuven. Lecture Notes in Computer Science, vol. 7428, pp. 233–250. Springer (2012)
10. Fisher, R.A.: Statistical Methods for Research Workers. Oliver and Boyd, Edinburgh (1925). http://psychclassics.yorku.ca/Fisher/Methods/
11. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: 10th International Workshop on Cryptographic Hardware and Embedded Systems – CHES 2008, Washington, DC. Lecture Notes in Computer Science, vol. 5154, pp. 426–442. Springer (2008)
12. Guilley, S., Hoogvorst, P., Pacalet, R.: Differential power analysis model and some results. In: Kluwer (ed.) Proceedings of WCC/CARDIS, Toulouse, pp. 127–142 (2004). doi:10.1007/1-4020-8147-2_9
13. Heuser, A., Rioul, O., Guilley, S.: A Theoretical study of Kolmogorov-Smirnov distinguishers – side-channel analysis vs. differential cryptanalysis. In: COSADE, pp. 9–28 (2014). http://dx.doi.org/10.1007/10.1007/978-3-319-10175-0_2

14. Heuser, A., Kasper, M., Schindler, W., Stottinger, M.: How a symmetry metric assists side-channel evaluation – a novel model verification method for power analysis. In: Proceedings of the 2011 14th Euromicro Conference on Digital System Design (DSD '11), Oulu, pp. 674–681. IEEE Computer Society, Washington, DC, (2011). doi:10.1109/DSD.2011.91. http://dx.doi.org/10.1109/DSD.2011.91
15. Kardaun, O.: Classical Methods of Statistics. Springer, Berlin/New York (2005)
16. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of CRYPTO'99, Santa Barbara. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999)
17. Le, T.H., Berthier, M.: Mutual information analysis under the view of higher-order statistics. In: Echizen, I., Kunihiro, N., Sasaki, R. (eds.) IWSEC, Kobe. Lecture Notes in Computer Science, vol. 6434, pp. 285–300. Springer (2010)
18. Maghrebi, H., Rioul, O., Guilley, S., Danger, J.L.: Comparison between side channel analysis distinguishers. In: Chim, T.W., Yuen, T.H. (eds.) ICICS, Hong Kong. Lecture Notes in Computer Science, vol. 7618, pp. 331–340. Springer (2012)
19. Mangard, S., Oswald, E., Standaert, F.X.: One for all – all for one: unifying standard DPA attacks. Inf. Secur. IET **5**(2), 100–111 (2011). ISSN: 1751-8709; doi:10.1049/iet-ifs.2010.0096
20. Moddemeijer, R.: On estimation of entropy and mutual information of continuous distributions. Signal Process. **16**(3), 233–248 (1989). http://www.sciencedirect.com/science/article/B6V18-48V26YR-MK/1/47d01a088dc7fbf6882c73ec582c81a2
21. Moradi, A., Mousavi, N., Paar, C., Salmasizadeh, M.: A comparative study of mutual information analysis under a Gaussian assumption. In: WISA (10th International Workshop on Information Security Applications), Busan. Lecture Notes in Computer Science, vol. 5932, pp. 193–205. Springer (2009)
22. Prouff, E.: DPA attacks and S-boxes. In: FSE, Paris. Lecture Notes in Computer Science, vol. 3557, pp. 424–441. Springer, (2005). http://www.springerlink.com/
23. Prouff, E., Rivain, M.: Theoretical and practical aspects of mutual information based side channel analysis. In: Springer (ed.) ACNS, Paris-Rocquencourt. Lecture Notes in Computer Science, vol. 5536, pp. 499–518 (2009)
24. Prouff, E., Rivain, M.: Theoretical and practical aspects of mutual information-based side channel analysis. Int. J. Appl. Cryptogr. (IJACT) **2**(2), 121–138 (2010)
25. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)
26. Rao, C.R.: Linear Statistical Inference and its Applications, 2nd edn. Wiley, New York (1973)
27. Rivain, M.: On the exact success rate of side channel analysis in the Gaussian model. In: Selected Areas in Cryptography, Sackville. Lecture Notes in Computer Science, vol. 5381, pp. 165–183. Springer (2008)
28. Rogaway, P. (ed.): Proceedings of the Advances in Cryptology – CRYPTO 2011 – 31st Annual Cryptology Conference, Santa Barbara, August 14–18, 2011. Lecture Notes in Computer Science, vol. 6841. Springer (2011)
29. Rudin, W.: Principles of Mathematical Analysis, 3rd edn. International Series in Pure and Applied Mathematics. McGraw-Hill, New York (1976).
30. Saon, G., Padmanabhan, M.: Minimum Bayes error feature selection for continuous speech recognition. In: Leen, T.K., Dietterich, T.G., Tresp, V. (eds.) NIPS, Denver, pp. 800–806. MIT (2000)
31. Silverman, B.W., Green, P.J.: Density Estimation for Statistics and Data Analysis. Chapman and Hall, London (1986)
32. Standaert, F.X.: Introduction to side-channel attacks secure integrated circuits and systems. In: Verbauwhede, I.M.R. (ed.) Secure Integrated Circuits and Systems. Integrated Circuits and Systems, chap. 2, pp. 27–42. Springer, Boston (2010). doi:10.1007/978-0-387-71829-3_2. http://dx.doi.org/10.1007/978-0-387-71829-3_2
33. Standaert, F.X., Bulens, P., de Meulenaer, G., Veyrat-Charvillon, N.: Improving the Rules of the DPA Contest. Cryptology ePrint Archive, Report 2008/517 (2008). http://eprint.iacr.org/2008/517

34. Standaert, F.X., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: EUROCRYPT, Cologne. Lecture Notes in Computer Science, vol. 5479, pp. 443–461. Springer (2009)
35. Standaert, F.X., Peeters, É., Rouvroy, G., Quisquater, J.J.: An overview of power analysis attacks against field programmable gate arrays. Proc. IEEE **94**(2), 383–394 (2006). (Invited Paper)
36. Tchebichef, P.: Des valeurs moyennes. Journal de mathématiques pures et appliqués **12**(2), 177–184 (1867)
37. Thillard, A., Prouff, E., Roche, T.: Success through confidence: evaluating the effectiveness of a side-channel attack. In: Bertoni, G., Coron, J.S. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2013, Santa Barbara. Lecture Notes in Computer Science, vol. 8086, pp. 21–36. Springer (2013)
38. Veyrat-Charvillon, N., Standaert, F.X.: Mutual information analysis: how, when and why? In: Clavier, C., Gaj, K. (eds.) CHES, Lausanne. Lecture Notes in Computer Science, vol. 5747, pp. 429–443. Springer (2009)
39. Veyrat-Charvillon, N., Standaert, F.X.: Generic side-channel distinguishers: improvements and limitations. In: Rogaway (ed.) Proceedings of the Advances in Cryptology – CRYPTO 2011 – 31st Annual Cryptology Conference, Santa Barbara, August 14–18, 2011. Lecture Notes in Computer Science, vol. 6841, pp. 354–372. Springer (2011)
40. Whitnall, C., Oswald, E.: A Comprehensive Evaluation of Mutual Information Analysis Using a Fair Evaluation Framework. In: Rogaway, P. (ed.) Proceedings of the Advances in Cryptology – CRYPTO 2011 – 31st Annual Cryptology Conference, Santa Barbara, August 14–18, 2011. Lecture Notes in Computer Science, vol. 6841, pp. 316–334. Springer (2011)
41. Whitnall, C., Oswald, E.: A fair evaluation framework for comparing side-channel distinguishers. J. Cryptogr. Eng. **1**(2), 145–160 (2011)
42. Whitnall, C., Oswald, E., Mather, L.: An exploration of the Kolmogorov-Smirnov test as a competitor to mutual information analysis. In: E. Prouff (ed.) CARDIS, Leuven. Lecture Notes in Computer Science, vol. 7079, pp. 234–251. Springer (2011)
43. Whitnall, C., Oswald, E., Standaert, F.X.: The Myth of Generic DPA... and the Magic of Learning. Cryptology ePrint Archive, Report 2012/256 (2012). http://eprint.iacr.org/2012/256
44. Zhao, H., Zhou, Y., Standaert, F.X., Zhang, H.: Systematic Construction and Comprehensive Evaluation of Kolmogorov-Smirnov Test based Side-Channel Distinguishers. Cryptology ePrint Archive, Report 2013/091 (2013). http://eprint.iacr.org/2013/091

Chapter 11
Wireless Sensor Networks: Routing protocol for Critical Infrastructure Protection

Apostolos Leventis and Konstantinos Papadopoulos

Abstract In this chapter we present a routing protocol suitable for using Wireless Sensor Networks in critical infrastructure protection applications (e.g. industrial facilities, borders, pipelines etc.). Since network nodes may extend in a straight line for several kilometers, this linear topology of nodes poses special restrictions to network design. An architecture optimized to support linear Wireless Sensor Networks is described hereinafter. Simulation results outline the network quality features offering a dependable solution, while implementation using off-the-shelf components demonstrates the remarkable characteristics of sensor nodes in terms of power consumption.

11.1 Introduction

Wireless Sensor Networks (henceforth WSNs) are widely used in monitoring applications, for handling data produced by a variety of sensor devices. Their flexibility allows covering a diverse set of applications, such as environmental, industrial, automotive, civil constructions structural health monitoring etc. Placement of sensor nodes is a critical issue for network operation as it directly affects network management, operation and lifetime. Therefore, placement has to be considered for node installation, packet routing and node power consumption issues respectively. Usually monitoring applications are addressed by utilizing wired technology, employing copper or fiber-optic cables. However, use of cable use may not suit all kinds of applications, as cables are not always easy to install or may need to be hidden, while the weight of cabling itself may be a constraining issue.

Although the usual node placement paradigm assumes that sensor nodes are distributed over an area (either uniformly or randomly), there is a class of applications where linear distribution of sensor nodes is required. These applications include monitoring of border lines, perimeter of industrial facilities, underground mines, road highways, structural health of pipelines or bridges etc. All these applications

A. Leventis (✉) • K. Papadopoulos
Hellenic Aerospace Industry Tanagra P.O. Box 23, GR 320 09, Schimatari, Greece
e-mail: leventis.apostolos@haicorp.com; papadopoulos.konstantinos@haicorp.com

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_11

require nodes to be placed in a line, where sensors monitor signals of interest. This line can be either open or closed and extend from hundreds of meters up to several kilometers.

The placement constraint imposed by the topology has direct impact to network operation. In a linear topology, network nodes need to relay data towards a single concentration point. Hence failure of one or more intermediate nodes can lead to formation of isolated parts in the network and in some cases even cause network collapse. A failure can be due to either physical reasons (e.g. power extinction, node malfunction) or to an external attack that deactivates some nodes.

Contrary to a meshed topology that would allow alternative routes across the network to be established, in a linear topology routes are mostly fixed. This can cause nodes being close to the concentration point get more network load compared to those farther away, resulting to power consumption imbalances among nodes. Hence special techniques and routing schemes have to be developed in order to avoid any power imbalances among nodes and also guarantee the existence of communication link when an arbitrary number of nodes are lost. This strategy allows for reliable network setup and operation functions and ensures network dependability.

Despite their usefulness in a diverse set of applications, research on the area of linear wireless sensor networks is quite limited. The most significant scientific research papers attempting a theoretical approach towards the various aspects of linear WSNs are discussed in the following paragraph.

Some research on the performance of linear WSN performance has been done in [17] where the relationship between throughput and energy cost is analyzed for various types of WSNs, including linear ones. An approximate relation for connectivity probability in single-dimensional ad hoc networks is provided in [13], while [16] uses a queuing theory approach to study connectivity issues. A classification and characterization of various types of linear WSNs is provided in [14], while in [11] an attempt is made to develop a network architecture and protocol utilizing linear WSN structure. Energy consumption in relation to node placement is studied in [10], while in [18] a routing scheme having minimal energy requirements is discussed. In terms of applications utilizing linear WSNs, a possible use case in underground mines is described in [12], while in [15] security issues are outlined and a key pre-distribution scheme for linear WSNs is discussed.

The above literature tries to identify problems induced at various aspects of linear WSNs operation. The use cases analyzed refer to open structures, e.g. a bridge or a pipeline extending for tenths of kilometers. Till now, the case of a closed topology described hereinafter, such as a perimeter protection scheme, has not been investigated.

The specification of a protocol supporting linear wireless sensor network topologies is presented hereinafter, with the application example of its use in the case of a critical infrastructure (e.g. an industrial facility) protection system. This has been simulated using a network simulation tool and implemented using off-the-shelf components to verify its characteristics in terms of power consumption.

In the following, the network requirements for WSNs utilized in critical infrastructure protection applications are described and a brief description of the routing protocol developed is provided, covering the basic points of network setup and operation. Then, protocol simulation results are shown, as well as power consumption measurements for the implementation of a node using COTS components.

11.2 Requirements for WSNs in Critical Infrastructure Protection Applications

Critical Infrastructure Protection (CIP) is an area where employment of WSNs can offer significant advantages. In CIP applications a virtual "fence" is created around the area to be monitored by suitably placing network nodes equipped with various sensors devices (e.g. PIR, accelerometers, microphones, geophones etc.). Acquired data can be processed locally up to some extent and then forwarded to a central station.

The linear placement of nodes across the area being monitored has direct impact to network operation. Namely, it poses restrictions in terms of inter-node communication, sensor data aggregation and exchange of sensed data or control commands [11]. Therefore the network architecture needs to ensure increased reliability with minimal end-to-end communications delay. Furthermore, the requirements for increased network lifetime, QoS and security should also be considered.

Among the inherent characteristics of linear networks the most important ones, affecting network architecture, are summarized below:

- Long distance of the network, as the monitored area or structure can extend to several kilometers
- Fixed node position (no mobility)
- Restricted number of neighbors for each node
- Node location awareness, allowing nodes to have known and specific neighbors
- Well known communication pattern, since the direction of data flow is fixed.

The one-dimensional linear sensor network operates under severe resource constraints, including restricted energy supply, low bandwidth, scarce computational power and limited communication capability. To effectively design the appropriate architecture of the chain-type network, the following issues must be taken into account:

1. Develop a scalable and energy efficient networking architecture.
2. Develop energy efficient communication protocols.
3. Develop a MAC protocol that matches the unique topology of linear wireless sensor networks and enables seamless communication among sensor nodes.

The length of a linear WSN prohibits each sensor node from communicating directly with a central data concentrator. Therefore, to efficiently transport sensor

data to a central data station, implementation of a relay mechanism is required. As the directional transmission along the chain of sensor nodes will create significant latency (especially if the chain is long), to control the overall data latency the number of hops needs to be limited.

11.3 Routing Protocol Description

To fulfill the application requirements, a three-tier hierarchical architecture was defined. The lowest level consists of sensing elements, called Sensor Nodes (SNs). SNs are grouped into clusters, where a special node, Base Station (BS) acts as the Cluster Head (CH). Base Stations comprise the second level of hierarchy, forming a backbone that gathers data from all SNs and (after performing any aggregation functions) forwards them to the Network Control Center where the Network Coordinator (NC) is located. NC forms the upper level of hierarchy. Figure 11.1 outlines the architecture concept of the three-level linear wireless sensor network.

Typically the range of a CH is much larger than that of a SN. This allows the network to remain functional if one CH is lost due to reasons such as malfunction, power extinction or external attacks.

11.3.1 Upper Network Layer (Backbone) Formation

The network formation starts with backbone initialization. As soon as network nodes power up, each CH associates with a neighboring CH, starting from NC and up to those CHs that are most distant from NC. The purpose of this process is to

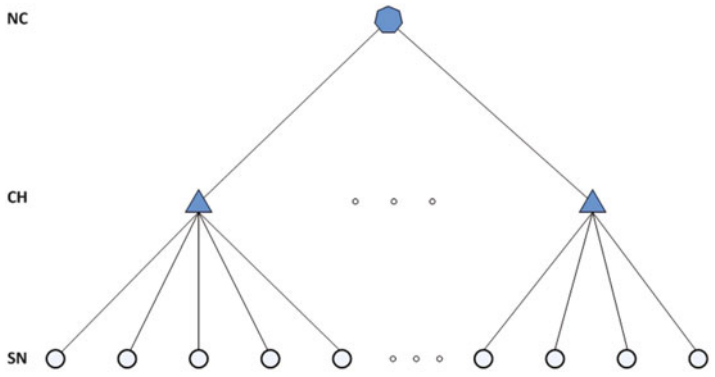


Fig. 11.1 Routing protocol architecture concept

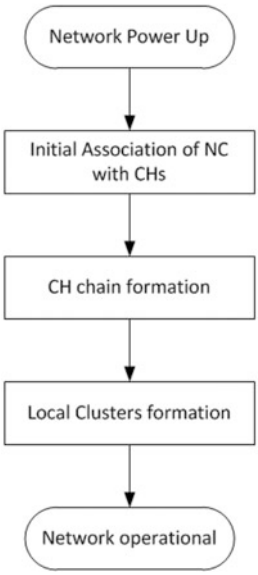


Fig. 11.2 Network initialization process

establish a route from each CH towards the NC and vice-versa. After all CHs have associated with another CH (or the NC), formation of local clusters starts in order to have each SN associate with a CH. This process is depicted in Fig. 11.2.

To compensate the low range of the CHs and provide increased connectivity, the distance between successive CHs is such that each CH can reach two neighboring CHs at each direction. Hence, the CH backbone connectivity is lost only if two consecutive CHs are lost.

A TDMA-based scheme is utilized in order to coordinate message passing among CHs towards NC and allow it happen during well-defined time slots. Timeslots are allocated to each CH by NC after the initial association and allow for data transmission in a timely manner. During their period a CH (or the NC) can request data from all nodes that it controls.

The beginning of each time slot is marked by a beacon transmitted by the CH (or the NC). After that, the active period follows, where all communications between CH and its children (SNs or CHs) take place.

Time slots must be distributed among CHs (including NC) in such a way that the network is fully covered; and also beacons from different CHs should not collide. The simplest way to achieve that would be to assign a unique time slot to each CH. However this would affect network scalability, since expanding the network would require additional time slots, causing degradation of network's performance.

By taking advantage of the network topology we can use the same time slot for CHs that are placed far away from each other, so that there is no interferance among them. This concept, called *time slot reuse*, is illustrated in Fig. 11.3. Since CH_5, CH_{10} and CH_{15} are far away from each other and can never have common children CHs, they can share the same time slot without causing any collisions.

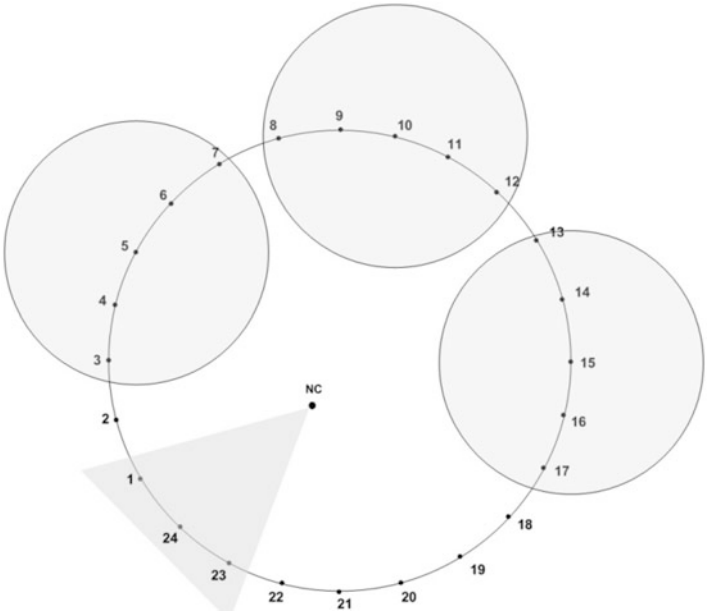


Fig. 11.3 Beacon time slot reuse concept

The arrangement of time slots among CHs can be such that seamless data transfer from sensors towards the NC is facilitated. By suitably arranging time slots among successive CHs, data packets can "slide" from the farthest CH to the NC with zero delay.

11.3.2 Lower Network Layer (Cluster) Formation

After the CH backbone is established and functioning, cluster formation, i.e. association of SNs with their nearby CHs can start. Backbone establishment implies that all CHs:

- Have already associated to each other in chain up to The network controller node (NC)
- Have received their respective beacon time slots from NC
- Have started transmitting their beacons in a timely manner according to their time-slot

By that time, all network's SNs are initialized and listen for beacons that can be heard at their location. As a result of this listening, every SN has built a list of nearby CHs and has chosen the CH having the strongest beacon signal as a candidate parent. Afterwards it wait the candidate parent to issue an association command.

Cluster formation is affected by sensor network topology and radio coverage issues. It must be noted that although the radio range of a CH is long enough to cover all SNs inside a cluster, the radio range of SNs is shorter. Hence, in some cases the CH can hear only those SNs that are close enough to it, while SNs at longer distances cannot be heard. On the other hand, all SNs will be able to hear messages transmitted by the CH. A relay mechanism must be utilized among SNs, so that SNs being closer to the CH relay messages from those SNs that are farther away.

11.3.3 Intra-cluster Communications

A network cluster is formed by a CH node and all SNs in its vicinity. SNs belonging to the same cluster hear the periodic beacons issued by the CH. The purpose of a beacon is twofold: to mark the beginning of cluster's active period and also to allow for clock synchronization of the cluster's SNs. All communications inside the cluster are curried out during the active period, as illustrated in Fig. 11.4.

Inside a cluster, the CH fully coordinates communications among nodes in a master/slave relationship. All communications within a cluster have the form of requests (or commands) issued by the CH to one or all its SNs and responses sent in turn from SNs to the CH. CH starts polling all SNs in the cluster for data by sending a separate message to each of them. Then, it waits their response for a reasonable time period which can depend on the number of hopes from the particular SN to the CH. In case of receiving an alarm (or receiving no response at all) the erroneous event will reported to the NC. In Fig. 11.5 request (DRQ_MSG) and response (DRSP_MSG) messages are shown for a two-hop data polling scenario.

Immediately after the CH has finished gathering data from its SNs, it places them into idle state even though the current active period has not ended yet in order to reduce their energy consumption.

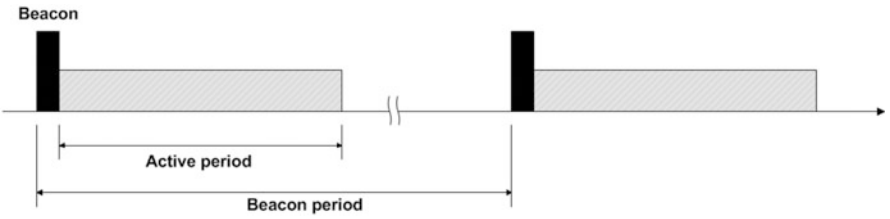


Fig. 11.4 Cluster's active period

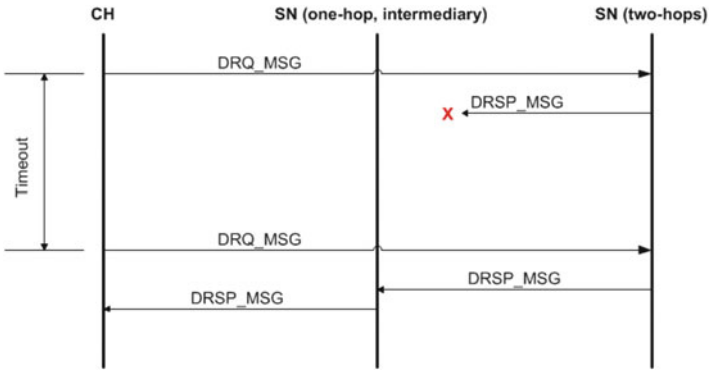


Fig. 11.5 Messages exchanged for a two-hop SN data polling

11.3.4 CH Backbone Communications

Data that a CH collects from its SNs as well as those originating from other CHs are transferred to NC.

During its active period, each CH must perform data communications with all the nodes it controls. For this, the CH initially communicates with all SNs that belong to its cluster to collect their data and then with its children CHs (if any) to get any data packets they have generated themselves and/or data packets they have already received from their own children CHs, if any.

By aggregating the collected data as they transverse the backbone, CHs relay them to the NC.

11.3.5 Network Recovery

11.3.5.1 Recovery from CH Failure

When a particular CH fails, malfunction is caused on both backbone and local cluster level. On the backbone level, if the failed CH has one or more children CHs, the association chain for those CHs and their children will be broken and will not relay their data towards NC. On the cluster level, all SNs reporting to the failed CH will become "orphans". For these reasons a trouble recovery procedure is necessary that will attempt to fix the problem without restarting the whole network.

Considering backbone, the failure of a CH will be noticed by its parent CH as the latter will not receive any response to its requests. Moreover, children CHs (if any) will stop hearing their parent's beacon.

Recovery from a CH failure depends on the backbone topology that has formed at a given time. Figure 11.6 illustrates two examples of CH failure in a closed topology consisting of 24 CHs.

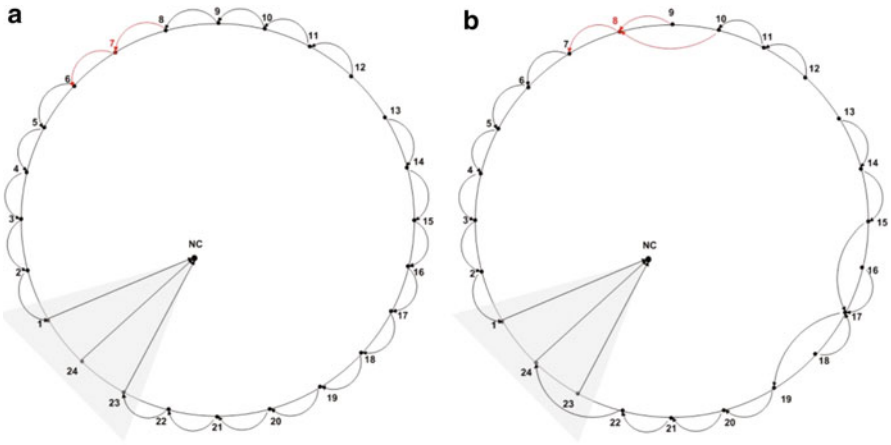


Fig. 11.6 Examples of CH failure impact

In topology (a), failure of CH_7 can be easily overcome by having CH_8 associate to CH_6 as child, while the (orphan) children SNs of CH_7 are shared between CH_6 and CH_8. In topology (b) however, CH_{10} cannot associate to CH9 as child, so loss of CH_8 means that both CH_9 and CH_{10} become orphans. In that case the situation can be resolved by having CH_{12} associate to CH_{13} as child, thus causing CH_{10}-CH_{12} to change their relay direction. Such functionality is supported by the protocol described herein.

11.3.5.2 Recovery from SN Failure

The most probable reasons of SN failure are hardware failure and battery discharge. Failure of a particular SN will be detected by its parent CH and will be reported to the NC. The network operator can then initiate procedures for SN repair or replacement. When the hardware problem is fixed and SN is installed and ready for normal operation again, it will select a CH and join into the relevant cluster.

11.4 Protocol Simulation

11.4.1 The CNET Network Simulator

The CNET network simulator [9] was used to simulate the functionality and assess the performance of the linear WSN network architecture described above. CNET is a network simulation tool developed by the University of Western Australia. It requires network protocols to be written in the C programming language

(C99 standard) and supports their execution within a single Unix-type process. A standard C compiler (e.g. gcc) is used to compile the user-written protocol code which is then dynamically linked with the CNET simulator at run-time.

The network topology is described via a topology file that defines nodes, links and attributes of the simulation. Via the topology file, characteristics such as node location, transmission frequency and power, antenna gain, radio sensitivity, current consumption etc. are specified. With reference to the ISO/OSI network model, CNET provides the Application and Physical layers. User-written protocols are required to "fill-in" any necessary internal layers and, in particular, overcome the corrupted and lost frames that CNET's physical layer randomly introduces.

11.4.2 Simulation Scenarios

For examining the protocol behavior, two scenarios were identified and simulated. The first scenario applies to closed line critical area infrastructure protection schemes, while the second one is better suited to open-ends applications such as pipelines, border monitoring etc. The simulation parameters were set according to the operational characteristics of the XBee radio transceiver [1] utilized in our test platform.

Scenario A The topology comprises of 24 Base Station nodes, arranged across the four sides of a square. As illustrated in Fig. 11.7 at the top left side is the Network Coordinator (NC), while the Sensor Nodes are not visible for illustration purposes but are uniformly placed between CHs. The distance among nodes is 200 m.

During network setup, the CH nodes identify the nearest neighbors in order to build the network backbone. Due to topology, two paths are created: one towards the right of CH_1 and the other towards the bottom of CH_{24}, as shown in Fig. 11.8.

After all CH nodes have been associated and received their slot information, the Sensor Nodes can then be queried for alarm events. This is illustrated in Fig. 11.9.

Figure 11.9 presents the data flow and also demonstrates time slot reuse: at the moment of the screenshot, three clusters, whose nodes are far away from each other, perform communication during the same time slot in order to accelerate error reporting towards the NC.

Scenario B The topology consists of 1 NC, 20 CHs and 8 SNs per CH populated in a single row. The distance among the CHs is 200 m while the SNs are equispaced between the CHs. Figure 11.10 shows the network layout, while the SNs are not visible.

During the network setup procedure the backbone is created, directing from CH_{20} to NC. After all CH nodes have been associated and received their slot information, Sensor Nodes can be queried for alarm events, as shown in Fig. 11.11.

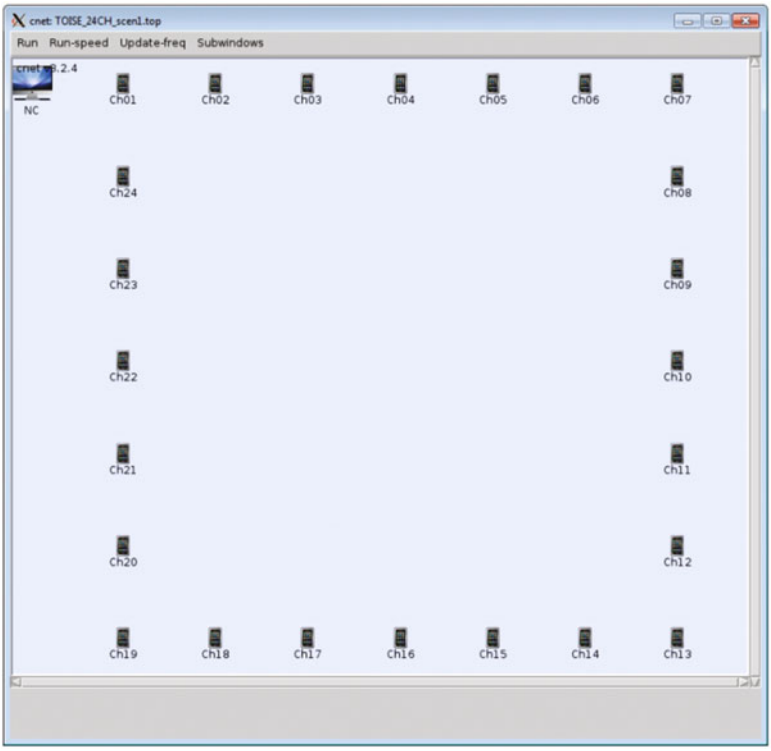


Fig. 11.7 Layout of upper network layer – closed topology

11.4.3 Simulation Results

In the following paragraph the outcome of the simulation results is provided. Although the two scenarios did not have exactly the same characteristics (such as number of CHs and SNs per cluster) due to restrictions of the simulator, the results are still valid and show network behavior.

A first conclusion derived by the simulation results is that the slot mechanism incorporated, in combination to the Request/Response messaging scheme and the utilized CSMA/CA protocol, allowed for almost collision-free network operation. As shown in Table 11.1, the reported collisions are about 1 % in both cases.

The slot mechanism has a direct impact to packet delay too. An efficient time-slot allocation scheme is vital and allows for quick error reporting to the NC, since data reports can transverse the CH backbone without additional delay. Table 11.2 summarizes the response time of an error event generated at the furthest end of each topology examined.

Clearly CH is the most sensitive type of the network nodes: not only does it query all SNs for data but it also relays other CHs' data towards the NC. The network

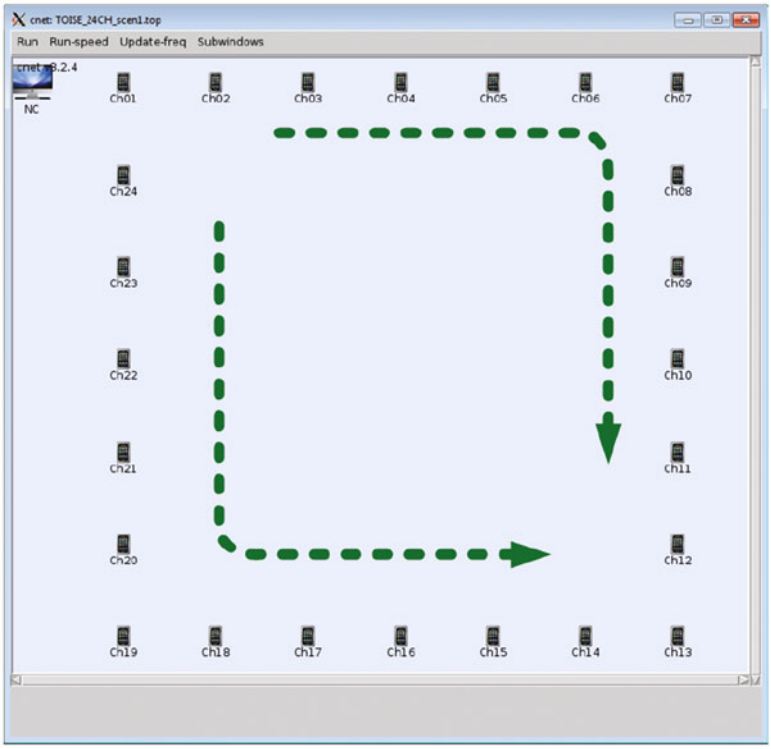


Fig. 11.8 Network backbone setup – closed topology

dependability is strengthened by utilizing CHs with overlapping range. Therefore, if one CH is compromised (either by an adversary attack or by a malfunction) the neighboring CHs will compensate and update their routing table so as to keep the network functional and will also inform the NC about the issue.

11.5 Hardware Implementation

For the hardware implementation of the linear WSN protocol, the STM32F103 Cortex-M3 microcontroller by STMicroelectronics [7] was utilized. The objective of hardware implementation was (a) to examine the Lower Network Layer functionality and (b) verify the power consumption characteristics of Sensor Node devices. Our study is limited to Sensor Nodes since in a network realization their population outnumbers Cluster heads' and consequently they have direct impact to the total power consumption of the network.Furthermore, since they need to be autonomously powered, their power-budget greatly affects the design.

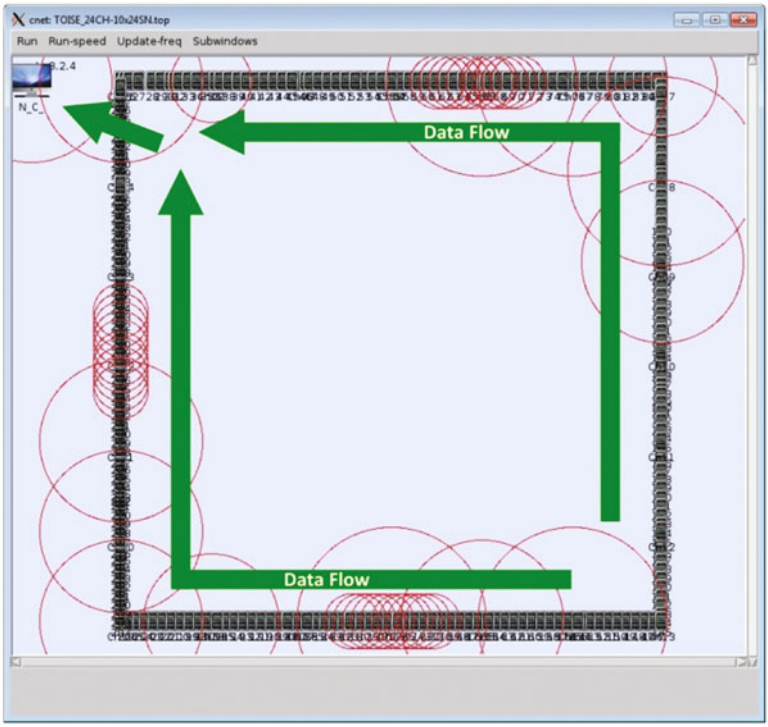


Fig. 11.9 Network operation – closed topology

Fig. 11.10 Network operation – open topology

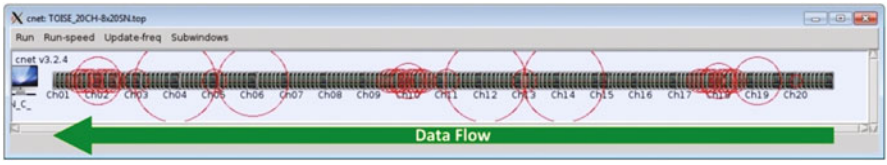


Fig. 11.11 Layout of upper network layer – open topology

Table 11.1 Frame transmission statistics

	Simulation time (s)	Transmitted frames	Number of CHs	Number of SNs	Reported collisions	Reported collision %
Scenario A	1,200	138,398	24	240	260	0.19
Scenario B	2,100	195,710	20	160	2,256	1.15

Table 11.2 Response time of error events

	Error event trigger time	Event reported to CH	Event reported to NC	Total of intermediate CH
Scenario A	t0	t0 + 1,001 ms	t0 + 4,000 ms	12
Scenario B	t0	t0 + 1,021 ms	t0 + 4.99 s	19

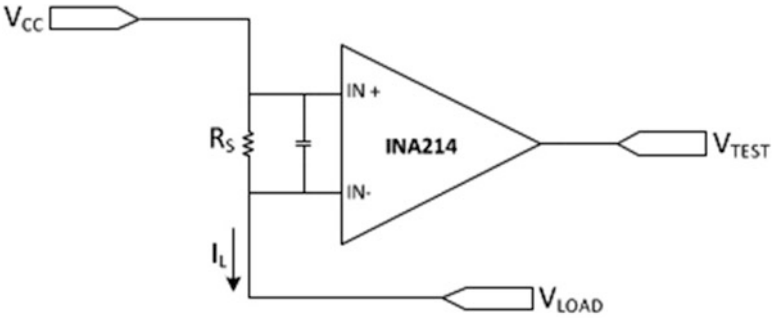


Fig. 11.12 Current monitoring subsystem

To allow MCU power consumption measurements, the off-the-shelf Olimex STM32-P103 evaluation board [4] was utilized and current measurement circuitry was added to its prototyping area as well as an IEEE802.15.4 [3] RF module and a PIR sensor.

A current measurement circuitry was built around the Texas Instruments INA214 current shunt monitor device [8], as shown in Fig. 11.12. The output voltage of INA214 is proportional to the current flowing through the resistor shunt (R_s). Two instances of the above circuit allow measuring separately the MCU and RF module current consumption.

The modified STM32-P103 board is shown in Fig. 11.13.

Regarding the application executed at the MCU, the ARM Cortex port of FreeRTOS real time kernel [2] was utilized. The CNET simulation model was then ported to FreeRTOS. Since CNET is an event-driven simulator utilizing interrupt handlers, timers and tasks to describe network functionality, porting the network model (written in C99) to FreeRTOS is quite a straightforward process.

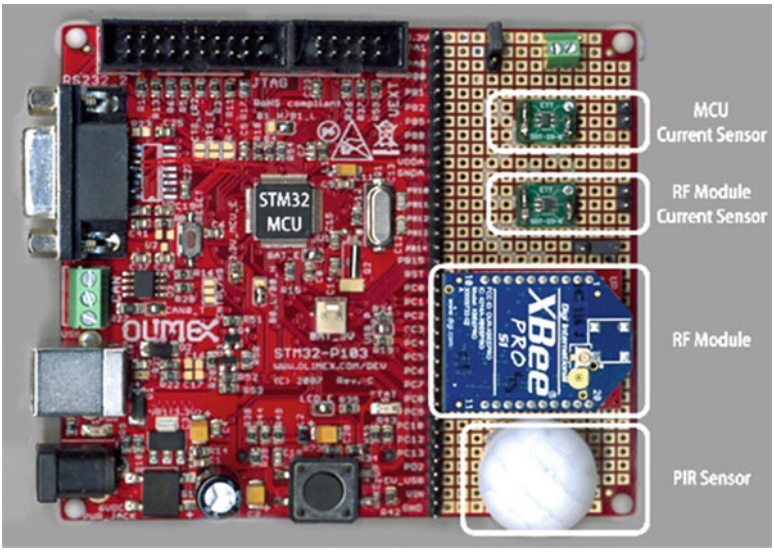


Fig. 11.13 Modified STM32-P103 board

11.5.1 Measurements

Measurements were made over the active period of the sensor node, during which the latter is operational and communicates with its Cluster head. Figure 11.14 shows the current consumption of both the MCU and RF module.

For each node, the maximum allocated timeslot duration is 250 ms but, as described earlier, the Sensor Node enters low-power state upon CH command, as soon as the later has successfully received the requested data. With reference to Fig. 11.14, early entering the low-power state reduces the active state to about 57 ms.

As indicated in Fig. 11.14, node operation can be divided to five distinct phases:

- MCU power up
- RF module power up
- Communications
- Shutdown
- Standby

These phases (whose duration is outlined in Table 11.3) repeat every 2,000 ms, which is the time period that the CH issues a new request to each Sensor Node for reporting any alarm events.

From the measurements is evident that the MCU current consumption is only a small fraction of the total node current consumption, while RF module is much more power-hungry.

Since all optimizations performed in the context of network architecture development refer to the network protocol being executed by the MCU and are independent

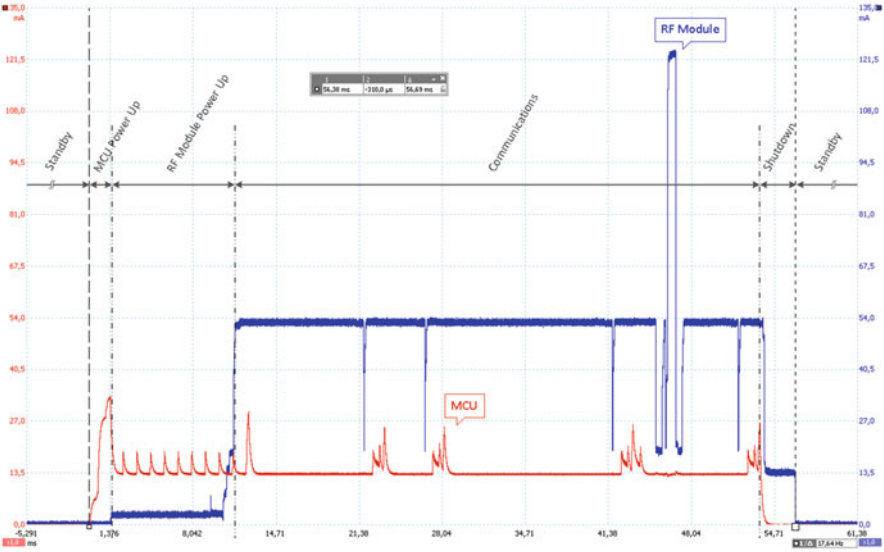


Fig. 11.14 MCU and RF Module current consumption

Table 11.3 Duration of MCU operation phases

Phases	Duration (ms)	Time %
MCU power up	1.81	0.09
RF module power up	9.75	0.49
Communications	42.32	2.12
Shutdown	2.81	0.14
Standby	1,943.31	97.17
Total1	2,000.00	100.00

of the RF module used, only the MCU consumption is considered in our study. Otherwise the results would be highly affected by the characteristics of the specific RF module used.

The above results show that the MCU of a SN device operates mostly in Standby mode. During standby, the current consumption is minimal (about 0.005 mA) as CPU, registers, memory and almost all peripherals are disabled.

The MCU current consumption during the active phases was measured to 14.02 mA, slightly above the MCU Sleep Mode current (about 9.5 mA according to the MCU datasheet [5]). Therefore the average current consumption, I_{avg}, is:

$$I_{avg} = \frac{I_{active} \times T_{active} + I_{standby} \times T_{standby}}{T_{active} + T_{standby}}$$

$$= \frac{14.02 \times 56.69 + 0.005 \times 1943.31}{2{,}000} \text{ mA} = 402.26\,\mu\text{A}$$

11.5.2 Results Analysis

The results provided in Sect. 11.5.1 were achieved by:

- Turning off all unused peripherals
- Utilizing the low power modes of the MCU (sleep mode whenever the O/S enters the IDLE state and standby mode whenever no communications are necessary)
- Designing the whole communications protocol with low power functionality in mind

Unfortunately it is not easy to define and/or calculate a strict metric identifying the reduction in current consumption due to the above optimizations, especially since some of them cannot be quantified, e.g. the effect of low-power architecture in the design of the network protocol. However, if we consider the extreme case of not using any low power mode at all, according to the MCU datasheet the current consumption would be about 31 mA. Therefore we can estimate the *MCU Current Reduction Ratio*, R as:

$$R = \frac{31\,\text{mA}}{402.26\,\mu\text{A}} = 77.06$$

i.e. reduction of about 77 times in our case.

The above comparison is illustrated in Fig. 11.15, where both data values are referenced to the MCU current consumption during Sleep Mode.

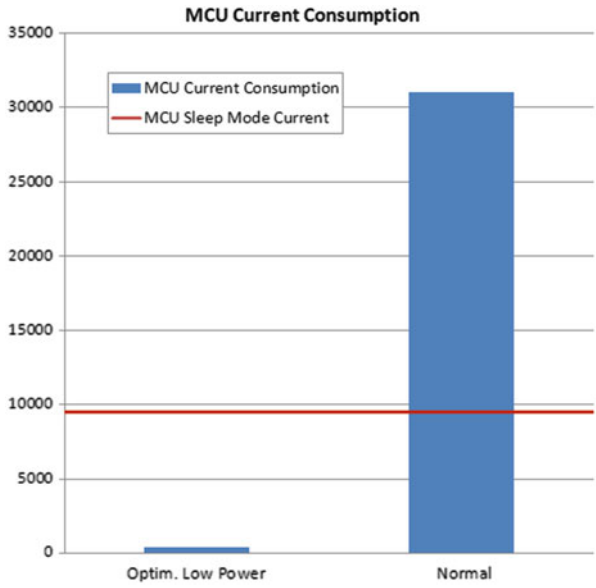


Fig. 11.15 MCU current consumption

Based on the above results, powering the MCU for a year would require a battery having capacity C of:

$$C = I_{avg} \times 365 \times 24\,\text{h} = 0.39983 \times 365 \times 24\,\text{mAh}$$

i.e. a 3,500 mAh battery would need to be used.

To provide a realistic view of the battery requirements that an actual sensor node could have, the RMS value of current consumed by both MCU and the XBee RF module was calculated. The consumption of these two parts would be dominant in a sensor node. The add-on current sensor circuitry was utilized to avoid measuring current consumption of unnecessary parts (such as leds, sensors etc.).

The calculation provided here is just for reference; an efficiently designed low power sensor node must be built around carefully selected low power hardware, tailored to the requirements of the specific application.

In Fig. 11.16 the three waveforms of current are shown.

The RMS value of total current during the active period was found to be 59.02 mA. Since the standby current for the MCU is 5 μA and the RF module 10 μA, the average current I_{TOT} is:

$$I_{TOT} = \frac{I_{active} \times T_{active} + I_{standby} \times T_{standby}}{T_{active} + T_{standby}} =$$

$$= \frac{59.02 \times 56.69 + 0.015 \times 1{,}943.31}{2{,}000}\,\text{mA} = 1.687\,\text{mA}$$

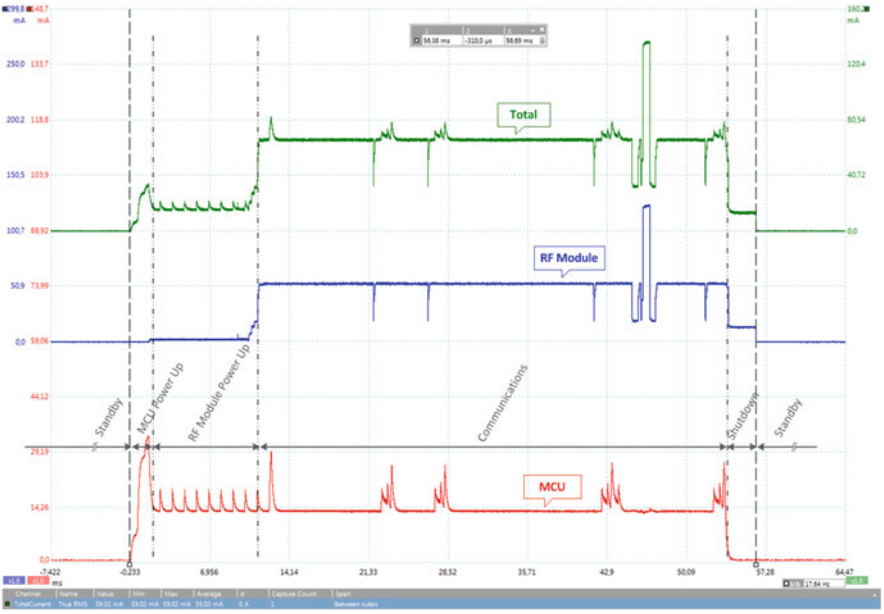


Fig. 11.16 MCU, RF module and total current

Based on the calculation, the 3,500 mAh battery considered above would last for:

$$t = \frac{3{,}500\,\text{mAh}}{1.677\text{mA}} = 2{,}074\,\text{h}$$

i.e. 2,074 h or about 3 months.

The above results are indicative of the power consumption levels that can be achieved using the proposed architecture. Still, a power-optimized product can achieve even better performance by using techniques such as lowering the MCU clock speed and utilizing optimized low-power devices which, for various reasons, were not available to use during prototype development. For example, the RF module used was found to be one of the most power-demanding devices of the market. Besides, an ultra-low-power MCU can be employed to replace the one of our prototype and achieve 50 % power reduction for the same clock frequency [6].

Conclusions
An architecture suitable to support linear Wireless Sensor Networks was described. Simulation results show its ability to offer a dependable solution for forwarding packets across a line extending up to several kilometers to a special node that acts as data sink for sensor data. Current measurements were performed on a prototype built using standard off-the-shelf components and demonstrate the remarkable power profile of sensor node devices.

Acknowledgements The authors would like to thank Mr. Aristeidis Nikologiannis for his valuable help on network simulation.

References

1. Digi International: Your M2M Expert. http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module. Website accessed 8 Sept 2014
2. FreeRTOS Real Time Operating System. http://www.freertos.org/. Website accessed 8 Sept 2014
3. IEEE 802.15 WPAN Task Group 4 (TG4). http://www.ieee802.org/15/pub/TG4.html. Website accessed 8 Sept 2014
4. Olimex STM32 Products. https://www.olimex.com/Products/ARM/ST/STM32-P103/. Website accessed 8 Sept 2014
5. STM32F103RB: Mainstream Performance line, ARM Cortex-M3 MCU with 128 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN. http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1031/LN1565/PF164487. Website accessed 8 Sept 2014
6. STM32L162RC: Ultra-low-power ARM Cortex-M3 MCU with 256 Kbytes Flash, 32 MHz CPU, LCD, USB, 2xOp-amp, AES. http://www.st.com/st-web-ui/active/en/catalog/mmc/FM141/SC1169/SS1295/LN13/PF253593. Website accessed 8 Sept 2014
7. STMicroelectronics. http://www.st.com/. Website accessed 8 Sept 2014

8. Texas Instruments Current Shunt Monitors. http://www.ti.com/product/ina214. Website accessed 8 Sept 2014
9. The CNET network simulator. http://www.csse.uwa.edu.au/cnet/. Website accessed 8 Sept 2014
10. Cao, M., Yang, L.T., Chen, X., Xiong, N.: Node placement of linear wireless multimedia sensor networks for maximum network lifetime. In: Proceedings of the 3rd International Conference on Advances in Grid and Pervasive Computing (GPC'08), Kunming, pp. 373–383. Springer, Berlin/Heidelberg (2008). http://dl.acm.org/citation.cfm?id=1788754.1788798
11. Chen, C.W., Wang, Y.: Chain-type wireless sensor network for monitoring long range infrastructures: architecture and protocols. Int. J. Distrib. Sens. Netw. **4**(4), 287–314 (2008). doi: 10.1080/15501320701260261
12. Chen, W., Sun, Y., Xu, H.: Clustering chain-type topology for wireless underground sensor networks. In: 8th World Congress on Intelligent Control and Automation (WCICA 2010), Jinan, pp. 1125–1129 (2010). doi: 10.1109/WCICA.2010.5554763
13. Ghasemi, A., Nader-Esfahani, S.: Exact probability of connectivity one-dimensional ad hoc wireless networks. Commun. Lett. IEEE **10**(4), 251–253 (2006). doi: 10.1109/LCOMM.2006.1613737
14. Jawhar, I., Mohamed, N., Agrawal, D.P.: Linear wireless sensor networks: classification and applications. J. Netw. Comput. Appl. **34**(5), 1671–1682 (2011). doi: 10.1016/j.jnca.2011.05.006. Dependable Multimedia Communications: Systems, Services, and Applications
15. Martin, K.M., Paterson, M.B.: Ultra-lightweight key predistribution in wireless sensor networks for monitoring linear infrastructure. In: Proceedings of the 3rd IFIP WG 11.2 International Workshop on Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks (WISTP'09), Brussels, pp. 143–152. Springer, Berlin/Heidelberg (2009). doi: 10.1007/978-3-642-03944-7_11
16. Miorandi, D., Altman, E.: Connectivity in one-dimensional ad hoc networks: a queueing theoretical approach. Wirel. Netw. **12**(5), 573–587 (2006). doi: 10.1007/s11276-006-6536-z
17. MomČilović, P., Squillante, M.S.: On throughput in linear wireless networks. In: Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'08), Hong Kong, pp. 199–208. ACM, New York (2008). doi: 10.1145/1374618.1374646
18. Zimmerling, M., Dargie, W., Reason, J.: Energy-efficient routing in linear wireless sensor networks. In: IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS 2007), Pisa, pp. 1–3 (2007). doi: 10.1109/MOBHOC.2007.4428618

Chapter 12
Wireless Sensor Networks: Virtual Platform for Performance Analysis and Attack Simulation

Álvaro Díaz, Javier González, and Pablo Sánchez

Abstract This chapter presents a virtual platform that is able to provide performance estimation of hardware/software systems, including communication networks (networked embedded systems). The framework is mainly focused on performance analysis of Wireless Sensor Networks (WSN) before deployment. The platform can also simulate the most common attacks that a WSN can suffer. This combination of unique features makes the virtual platform a key tool for simulating security in low power WSN at the first stages of the design process.

12.1 Introduction

Every WSN has specific nodes that must comply with system, network and sensor requirements. During last years, a common requirement of WSN specifications is security assessment. An attacker can have direct access to the network nodes and there are a huge number of possible ways to attack the node/network.

The required security level may vary depending of the importance of the data that is being obtained and/or exchanged. For this reason, it is crucial to identify the security weaknesses of the WSN at the design phase. Furthermore, understanding the potential effects of the most typical attacks on a node (or the entire network) helps to prevent problematic vulnerabilities. This is of great value when designing the node's embedded software and/or the complete WSN system.

Additionally, energy resources are also limited. Usually, power consumption is the greatest constraint in WSNs. Sensor nodes are commonly battery-powered devices, in which their useful-life depends on their battery life. These nodes are often placed in hostile environments with difficult node access which affects a possible battery replacement. Moreover, WSN attacks commonly produce an increase in the power consumption of the attacked nodes, with a reduction of their useful-life and a loss of communication performance.

Á. Díaz (✉) • J. González • P. Sánchez
University of Cantabria, ETSIIT, Avda. de los Castros, 39005 Santander, Cantabria, Spain
e-mail: adiaz@teisa.unican.es; javiergb@teisa.unican.es; sanchez@teisa.unican.es

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_12

Current WSN simulation tools do not offer the possibility to simulate typical attacks that these networks can suffer. The paper [7] presents an overview of current WSN simulation frameworks. NS-2 [10] and OMNET++ [11] are discrete event network simulators that have been used to model WSNs but without focusing in the security aspect. Another framework, TOSSIM [6], is a bit-level discrete event simulator and emulator of TinyOS [16].

As mentioned above, one of the main objectives of the attacks is to increase the power consumption of the node. To make an accurate estimation of the attack's effects, it is important to use a performance analysis framework that provides power consumption and execution time estimations. Some of the previously commented simulators can provide WSN simulation, Real-Time Operating System (RTOS) support and power/execution time estimation but, as far as the authors know, there is no simulation framework that integrates these features and attack simulation.

A virtual platform that integrates all the main features of WSN simulation frameworks (functional simulation, RTOS integration and performance analysis) and attack simulation is shown in this chapter.

12.2 Vulnerabilities in WSN

An attack can be defined as an attempt to gain unauthorized access to a service, resource or information. It could also be an attempt to compromise integrity, availability or confidentiality of a system [9].

The possible types of attacks are so high that is difficult to classify them. For instance, they can be focused in intercepting messages, corrupting and replaying them into the network or more sophisticated strategies such as in introducing a malicious node that behave as a fake gateway. Nevertheless, Mohammadi and Jadidoleslamy [8] classify the attacks into two big categories: passive and active attacks.

While passive attacks relate to privacy vulnerabilities as eavesdropping, gathering and stealing of the information by intercepting data communications or monitoring packets exchanged within a WSN, active attacks perform actions such as injecting faulty data into the WSN, impersonating, modifying resource and data streams, creating holes in security protocols, destroying sensor nodes, degrading performance, disrupting functionality and overloading the network. Table 12.1 shows the most important attacks that a WSN can suffer and what their nature is: passive or active.

Typical WSN attacks can be classified into 20 categories:

Jamming attack: This attack focuses in denying service to authorized users as legitimate traffic is jammed by the overwhelming amount of illegitimate traffic. It disrupts network functionality by broadcasting high-energy signals. Figure 12.1 illustrates how the Jamming attack behaves. The attacker or malicious node either produces large amounts of interference intermittently or persistently. This interference directly affects at least three network nodes (B, D and F).

Table 12.1 Active and passive WSN attacks

Physical layer	Jamming	Active
	Tampering	Passive
Link layer	Collision	Active
	Resource exhaustion	Active
	Energy drain	Active
	Interrogation	Active
	Sniffing	Passive
Network layer	Selective forwarding	Active
	Back hole	Active
	Sinkhole	Active
	Hello flood	Active
	Misdirestion	Active
	Sybil	Active
	Node replication	Active
	Spoofing	Passive
Transport layer	Desynchronization	Active
	Flooding	Active
Application layer	Homing	Active
	Path based DOS	Active
	Application	Active
	Overwhelm	Active

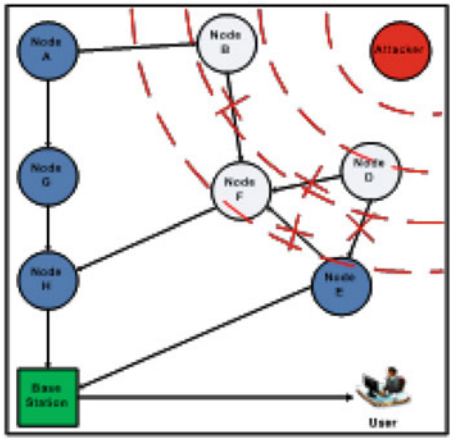


Fig. 12.1 Jamming attack

Tampering attack: Tampering attacks [4] require physical access to the node to steal its internal data. So, they can obtain sensitive information such as cryptographic keys.

Collision attack: In a collision attack [15], an attacker node does not follow the medium access control protocol and produces collisions with the neighboring node's transmissions by sending a short noisy packet. Packets collide when

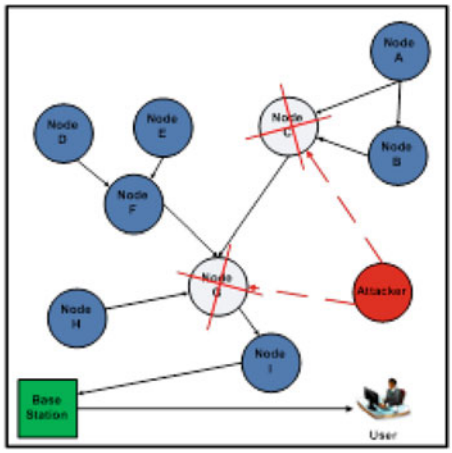


Fig. 12.2 Energy drain attack

two nodes attempt to transmit simultaneously on the same frequency, producing packet corruption. This attack can cause a lot of disruption to the usual network operation.

Resource exhaustion attack: Repeated collisions and multiple retransmissions are performed until the node cannot handle all the data or exhausts the battery. Malicious node continuously requests and/or transmits over the channel.

Energy drain attack: It is known of the difficulty of replacing sensor node batteries in WSN. So, attackers may use compromised nodes to inject false reports or to generate large amount of traffic in the network that will waste a lot of energy. The aim of this attack is to disconnect sensor nodes of the network, degrade network performance and, ultimately, take control of part of the sensor network by inserting a new sink node. Figure 12.2 shows how the attacker node generates false messages continuously. Its immediate neighbor nodes 'C' and 'G' responds to the attacker until the battery is emptied.

Interrogation attack: This attack exploits the two-way RTS (Request to Send)/CTS (Clear to Send) handshake that many MAC (Media Access Control) protocols use to mitigate the hidden-node problem. The attacker repeatedly sends RTS messages to obtain CTS responses from a targeted neighboring node.

Sniffing attack: Sniffing attacks consist in capturing network packets. Once the packet is captured using a sniffer, the contents of the packets can be analyzed and some private information could be stolen.

Black hole attack: The Black Hole attack basically consists in an alteration of the network routing with the objective of attracting all the packets to the attacked node destination. Figure 12.3 shows a Black Hole attack, where all the packets sent by the nodes D, E, G, and I are captured and dropped by the attacker.

Selective forwards attack: Multi-hop networks assume that the nodes will faithfully forward and receive messages. However a malicious node may refuse to forward certain messages and simply drop them, ensuring that they are not propagated any further. This is what a Selective Forward attack tries to

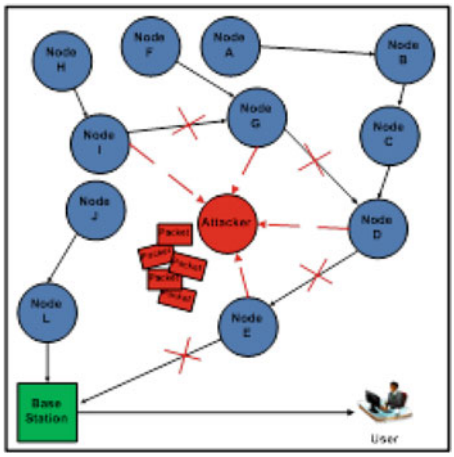


Fig. 12.3 Black hole attack

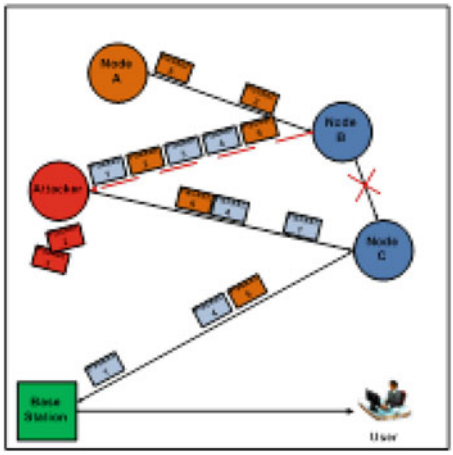


Fig. 12.4 Selective forward 1

accomplish. The procedure to launch Selective Forward attack is very similar to the Black Hole. First, a malicious node has to convene the network that it is the nearest node to base station, attracting network traffic to route data through it. Then, random packets (Fig. 12.4) or specific packets (Fig. 12.5) can be dropped.

Homing attack: In a Homing attack, the attacker observe the network traffic to deduce what is the geographic location of the most critical nodes, such neighbors of the base station. Then, the attacker tries to physically disable these nodes. This leads to another type of Black Hole attack.

Sing hole attack: When performing a Sink Hole attack, the adversary tries to attract all the traffic from a particular area through a compromised node. The compromised node is placed at the center of some area and creates a large "sphere of influence", attracting all traffic directed from the sensor nodes to a base station.

Hello flood attack: In a Hello flood attack, the attacker typically attempts to drain the energy from a node or exhaust its resources. An attacker with a large

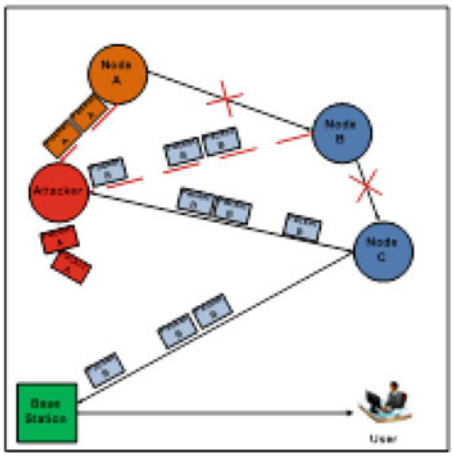


Fig. 12.5 Selective forward 2

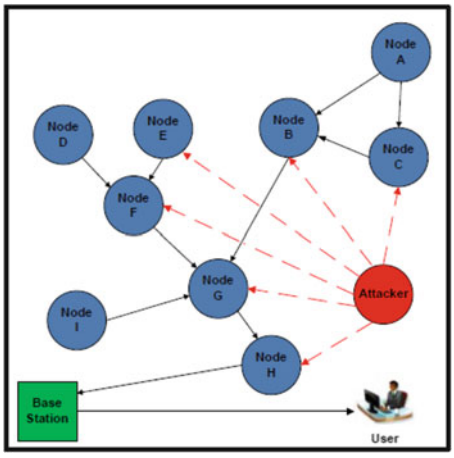


Fig. 12.6 Hello flood attack

transmission power could broadcast "HELLO" packets (used in many protocols for node discovery) to convince every node in the network that the adversary is within one-hop radio communication range. This causes a large number of nodes to waste energy by sending packets to this imaginary neighbor. Figure 12.6 depicts how an adversary node "Attacker" broadcast "hello" packets to convince nodes in the network that is trying to contact them.

Sybil attack: This attack consists in the modification of the network routing with the objective of isolating certain nodes. When these nodes can no longer communicate, the attacker sends fake traffic supplanting them.

Misdirestion attack: In this attack [1], the attacker routes the packet from its children to other distant nodes, but not necessarily to its legitimate parent. The main objective of the intruder is to misdirect the incoming messages to increase the latency, which prevents a few packets from reaching the base station.

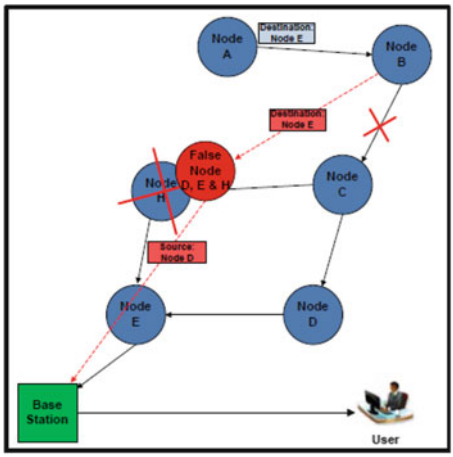


Fig. 12.7 Sybil attack

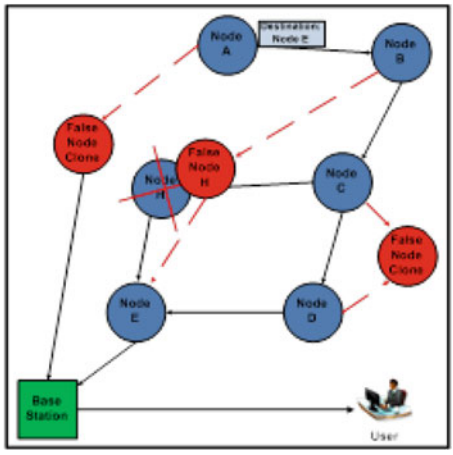


Fig. 12.8 Node replication attack

Node replication attack: This is an attack where the attacker tries to mount several nodes with the same identity at different places of the network. Although Sybil attack and Node Replication attack could seem similar, in Sybil attack a single node exists with multiple identities while in node replication attack multiple nodes are present with the same identity. Therefore, in Sybil attack an adversary can succeed by attacking only a single node whereas the node replication attack requires more nodes of the network. Figures 12.7 and 12.8 shows an example of node replication attack.

Spoofing attack: A spoofing attack is a situation in which the attacker successfully impersonates another by falsifying data and thereby gaining an illegitimate advantage. The idea of this attack consists in target routing information while it is being exchanged.

Flooding attack: An attacker may repeatedly make new connection requests until the resources required by each connection are exhausted or the maximum limit is reached.

DoS attack: In a path-based Denial of Service (DoS) attack, an adversary swamps sensor nodes at long distance by flooding a multihop end-to-end communication path with either replicated packets or spurious injected packets.

Application attack: This attack modifies the firmware or software node. The attacker directly accesses to the node and modifies the software code.

Overwhelm attack: An attacker might attempt to overwhelm sensor nodes with sensor stimuli, causing the network to forward a large number of packets to a base station.

In the simulation perform presented in Sect. 12.3, the virtual platform will model those attacks who performs malicious attempts to disrupt total or partially the communication flow within network nodes.

12.3 Virtual Platform Simulation Technique

What are the main components of the virtual platform and how to obtain performance estimation of a WSN are explained in this section. Next sections will focus in how the attacks and the cryptography security are modeled and evaluated.

12.3.1 Hardware/Software Co-simulation

Co-simulation is a fundamental part of the verification task in Hardware/Software (HW/SW) co-design flows. The most popular approach to integrate the embedded software in system co-simulation is the use of instruction-set processor models (ISS) to execute the binary code of the SW components. This solution is accurate but also too slow to be used at the beginning of the design process. Therefore, it is proposed to use the native simulation methodology which is faster than ISS approaches. With this technique, software application execution is simulated in a host computer using abstracts models of the processor, RTOS and hardware platform.

The co-simulation methodology described in this subsection is based on the native simulation approach [13] depicted in Fig. 12.9. This approach consists in a combination of the native execution of annotated software code with the use of a virtual platform model of the hardware architecture. With this simulation technique it is possible to model hardware platform components in System-C and execute the code of each node over this platform. Thus, it possible to obtain fast and accurate estimations of energy and power consumption, execution time, data a/instruction cache hits and misses, number of bus accesses, network traffic, etc.

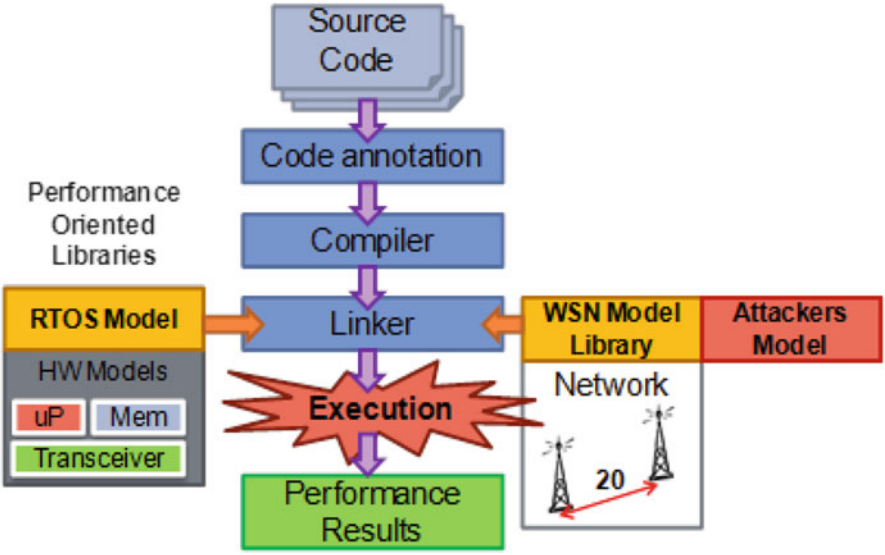


Fig. 12.9 Native co-simulation process in the WSN virtual platform

The framework (Fig. 12.9) also includes RTOS, network and the attacker model. The co-simulation process includes several steps:

- The embedded source code is parsed and analyzed. The basic blocks are identified and annotated with several performance-oriented parameters (energy consumption, execution time, cache and bus access requirements, etc.).
- The instrumented code is natively compiled and linked with several additional libraries that implement platform components or required models: processors, network, RTOS, bus, cache, etc. The execution of this code will produce the performance analysis results.

12.3.2 Wireless Sensor Network Model

Figure 12.10 presents an example of wireless sensor network in which the node architecture has been detailed. It can be observed that the architecture and application of each node could be independent from other nodes (heterogeneous network).

But it is still necessary to model the behavior of the network. In a wireless system, the physical channel between two nodes is a shared channel, with noise and interference. Additionally, the transmission range of the nodes is limited. As a consequence, WSN developers need to determine the node visibility and the probability of a successful reception of a packet. Thus, it is necessary to define a matrix with the probability of packet loss for the whole network. This probability

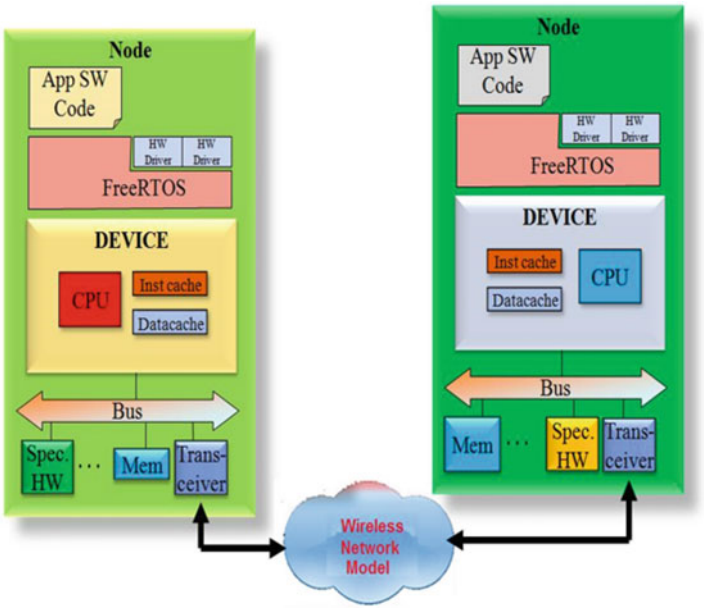


Fig. 12.10 Representation of the nodes architecture

data may be calculated by electromagnetic-propagation simulation tools such Cindoor [17]. With this matrix the simulator can estimate the effectiveness of the links between nodes.

A high level scheme of the network model is presented in Fig. 12.11. As it can be observed, the hardware network interface is responsible for deciding which packets should actually be received by the node. In a real wireless network, when a node sends a packet to another node, this packet is not only received by the receptor node but also by all the nodes in the transmission range of the sender. In this case, the hardware network interface is responsible for disposing of the packets that do not correspond to its node.

A packet-loss probability has to be defined for every pair of nodes. For example, a 100 % packet-loss probability means that the sending node range is not enough to directly reach the destination node, thus all the transmitted packets are lost. Instead, if the developer defines the link as 0 %, all the packets reach its destination. For example, the link between the node 0 and node 1 is defined as 10 % in the Fig. 12.12. Thus, it indicates that 90 packets reach its destination for each 100 packets transmitted.

It is important to clarify that the network is responsible for transmitting the packets to their destination. When a node sends a packet, the network adds the packet to the transmission queue that is sorted be the time of arrival at the reception node. When the simulation time matches the time of arrival of the packet,

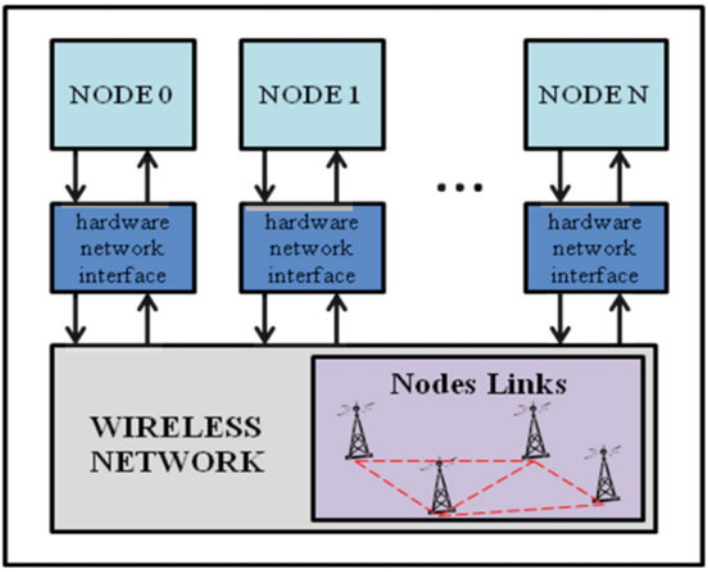
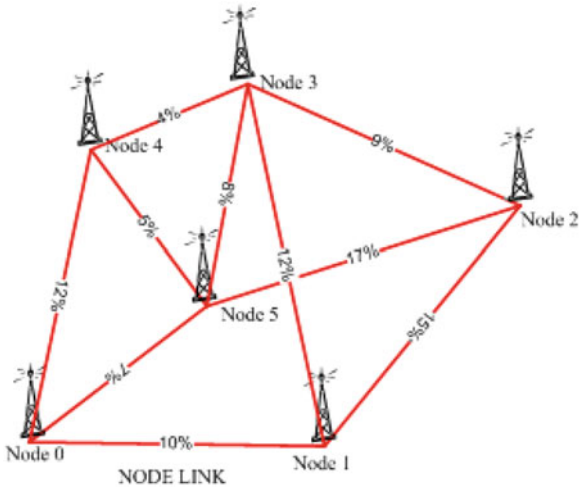


Fig. 12.11 Network model

Fig. 12.12 Wireless network model



the wireless network pops the packet and generates a real random number between 0 and 100. If the probability of success is higher than this random number, the network transmits the packet to the destination node; otherwise the packet is discarded. The probability of success, $P(success)$, is understood as 100 minus the probability of error, $P(error)$, that is defined in the packet-loss probability matrix. Figure 12.13 represents a scheme of this wireless network operation.

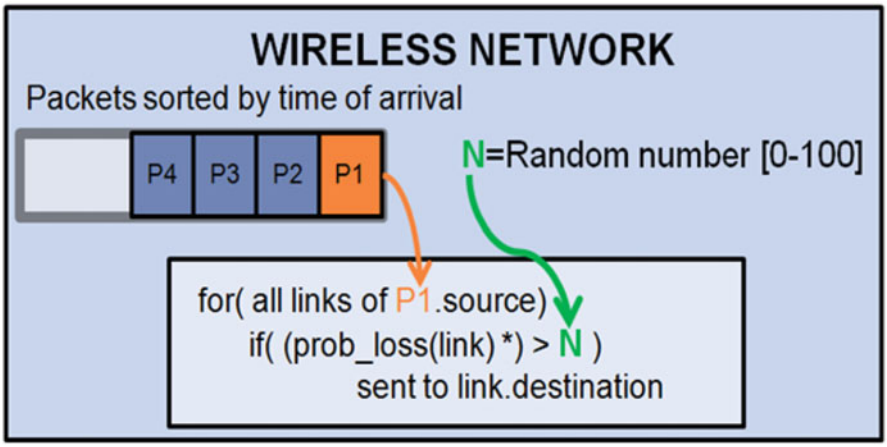


Fig. 12.13 Simulation of a wireless network

However, the transmission channel is shared in a real wireless network and a transmitted package is visible by all the nodes in the range. This is the reason why in the simulation the packets are transmitted not only to the destination node but to all the nodes in the range.

12.3.3 Modeling of the Node Hardware Elements

There are two essential components in a wireless sensor node that other systems do not have. These components are the sensors and the radio-frequency (RF) transceiver, thus it is necessary to model them adequately in the Virtual Platform.

The sensor is responsible for sampling external information with a certain frequency or when an event occurs. In the simulation, this component is implemented as an external component with specific power consumption. It is possible to simulate any type of sensors (temperature, movement, etc.). They only differ in the information that is sent to the node, its power consumption and operating frequency.

The other important component is the RF transceiver: The behavior of this device is more complex than the sensor. It is implemented with several configuration registers to control its operation. These registers can be modified dynamically in order to configure the transceiver operation. Software applications can access them through an API (Application Programming Interface) layer that implements the RF-transceiver AT commands. The current version of the framework models a commonly used RF transceiver, XBee [26], which includes network protocol management. In the evaluation scenarios, the XBee default configuration is used. For simulation, the most interesting transceiver parameters are:

- **Destination Address (High and Low)**: they define the address of the destination node for transmission.
- **Baud Rate**: speed for data transfer between radio transceiver and host.
- **MAC Retries**: number of retries that can be sent when transmitting.
- **Multiple Transmissions**: number of additional broadcast retransmissions.
- **Power Level**: power level at which the RF module transmits.

12.3.4 RTOS Model

It is also important to model other platform elements as the Real-Time Operating System (RTOS). Most WSN node applications require an embedded RTOS. This is an operating system that serves real-time application requests. The RTOS can create and operate a number of concurrent user tasks, scheduling those tasks within guaranteed time-limits using some priority mechanism. This guarantees that there is no-interference between these tasks and between the tasks and the kernel, allowing these tasks to communicate using queues, to synchronize using mutexes or semaphores and to interact directly with hardware through device drivers, clocks and interrupt mechanisms.

This RTOS model allows simulating correctly the system functions and achieving more accurate estimations. Additionally, this allows using of the same RTOS API that the physical platform uses, thus the same software code can be executed in the physical node and in the simulator.

The Virtual Platform runs natively POSIX [14]. This interface implements threads, mutexes, semaphores, queues, timers and other common POSIX services. These elements are needed for the RTOS model so the new RTOS layer uses these services provided by the POSIX layer.

One of the most popular RTOS in WSN is FreeRTOS [5]. In order to integrate this RTOS in the virtual platform, a FreeRTOS layer was created on top of the existing POSIX API. It is possible to simulate applications developed for this operating system because of this layer as it can be observed in the Fig. 12.14.

12.4 Attack Modeling Technique

Besides the WSN elements shown in the previous section, the virtual platform also allows the modeling of attacking nodes. These nodes represent the attackers with basic simulation models to allow a fast simulation. Three basic types of attacker nodes are defined: “Link-Noise node”, “Fake Packet Injection node” and “Direct Attack node”. The idea is that with these three basic types of attackers is possible to cover all the vulnerabilities defined previously.

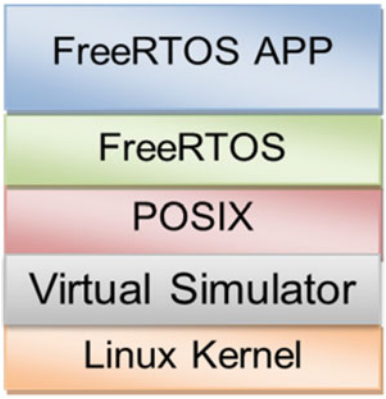


Fig. 12.14 FreeRTOS in the virtual simulator

12.4.1 Link-Noise Attacker

A Link-Noise node dynamically modifies the packet-loss probability between nodes (reduction of the communication link quality), thus some packets are incorrectly discarded. The link-noise attacker enables the definition of several parameters:

- **Links**: List of communication links or node-pairs that are affected by this attacking node.
- **Power**: List of noise power that will be applied to every defined link. This specifies the percentage of packet-loss probability that will be added to the original probability.
- **NumPackets**: Percentage of packets affected by the increased packet-loss probability.
- **Time**: It defines the ranges of time in which the attacking is performed. The attacker can be switched on or off many times during the simulation.
- **TypePackets**: The attack will only be active for specific packet types, enabling the simulation of selective attacks.

In order to simulate Link-Noise nodes efficiently, the previously commented WSN network model has been modified. Basically, this attack modifies the packet-loss probability for certain packet types during pre-defined periods of time. The network simulation model includes the new probability as shown in Fig. 12.15. When a packet has to be transferred to the receiver node, the reception probability will include the original probability and the additional probability noise caused by the attack ($prob\_loss$ in Fig. 12.15).

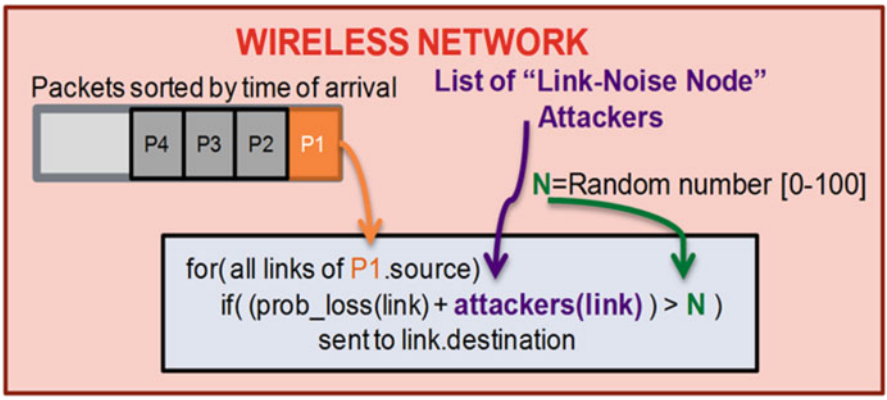


Fig. 12.15 Simulation with link noise attackers

12.4.2 Fake Packet Injection Node

This attacker introduces fake packets into the network with different undesired consequences. Fake packets are received by the nodes because their structure is formally correct. The packet payload depends of the attacker parameters that model the attack mode, injection frequency, fake packet types, etc. These parameters are:

- **Frequency**: It defines the fake packet rate or number of packets per second that are injected into the network.
- **TypePackets**: It defines the type of packets that the attacker injects. As for example: ACK, RTS and data packets.
- **Time**: It defines the time interval in which the attack is being executed.
- **NodesDestine**: It specifies the receiver nodes.
- **Broadcast**: Each packet will be sent to all nodes.

In order to simulate this attack, the attacker node injects new packets into the transmission queue of the network model. Once the packet is inserted in this queue, the network transmits these fake packets as if they were genuine. Figure 12.16 shows this attack in the network model.

12.4.3 Direct Attack Node

The Direct Attack modifies the embedded software of the node by a introducing a fake program. Usually, it is performed by downloading the undesired application to the node. This attack is modeled in the virtual platform by modifying the packet-loss probabilities of the network.

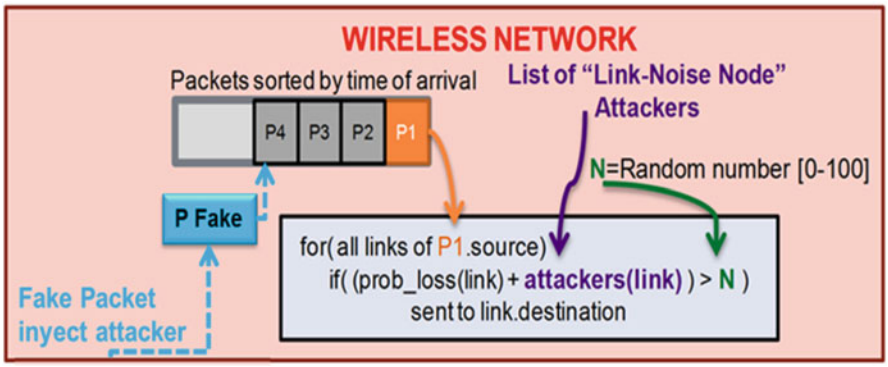


Fig. 12.16 Simulation with fake packet injection attackers

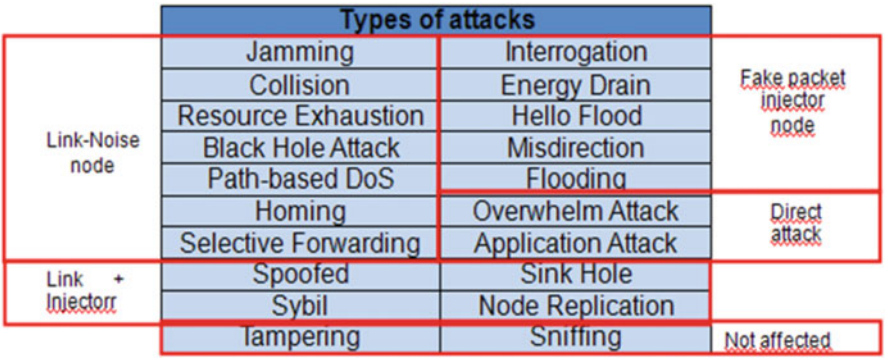


Fig. 12.17 Implementation nodes for each attack

12.4.4 WSN Attack Model and Its Relationship with the Identified Vulnerabilities

The relationship between the described attacks and the attacker model will be explained in this subsection. Figure 12.17 shows the relationship among the identified WSN attacks and the proposed attacker models. These models cover all the identified attacks explained before. For example, the Direct Attack node enables the modeling of the swamping and application attacks shown in the vulnerabilities section.

As it can be observed in Fig. 12.17, a Link-Noise node can model different attacks: Jamming, Collision, Black Hole and Path-based attacks. Different parameter values of the attacker node configuration enable modeling of different types of attacks. For example, jamming and collision attacks can be modeled with the same attacker node (Link-node) but with different parameters. These parameters are not only used to define attacks, but also to define attack strategies. For example,

the Jamming attack can have multiple strategies. If there is a PartialBand Noise jamming attack [12], the attacker could be modeled as a Link-noise with 50 % of affected packets.

Another example is the interrogation attack. It is simulated with a Fake-Packet injection node. In this case, it is necessary to configure the attacker node to continually send CST and RTS packets.

Sybil attack modeling is a special case because it is necessary to use two attacker nodes. The Link-Noise node isolates the attacked node and the Fake Packet injector node inserts the altered packets. The tampering and sniffing attacks are not simulated because they do not affect the operation of the node. These attacks are focused in stealing information.

12.5 Evaluation of WSN Attacks

12.5.1 Virtual Simulator Results

In order to use the Virtual Platform to simulate the SW application execution on an embedded HW some files have to be processed. First, the software applications, that have to be written in C/C++, have to be compiled with the virtual simulator compiler. In addition, several library elements that model the virtual platform components are required to describe the complete HW/SW system for each node. Besides this, the network definition file with the network packet-loss probabilities is required. Once all the input files have been processed, the virtual simulator generates an executable model. This model is used to simulate the complete system. After the execution is completed, the following performance estimations are provided:

- **RTOS related estimations**
  - Number of created processes
  - Number of destroyed processes
  - Mean execution time of a process
  - Total user time
    - Total kernel time
  - Number of thread switches per processor
  - Number of task switches per processor
- **RF-Transceiver and network estimations**
  - Transceiver energy consumption by transmissions in Joules
  - Transceiver power in Watts
  - Number of packets send
  - Number of packets received

- **Hardware related estimations**
  - Number of nano-seconds that the every simulated core have been executing code
  - Percentage of core use
    - Number of executed instructions
    - Number of instructions cache misses
    - Number of data cache misses
    - Core power and energy, in Watts and Joules respectively
    - Instruction cache power and energy, in Watts and Jules respectively
    - Data cache power and energy, in Watts and Joules respectively
- **Total simulation time in seconds**

12.5.2 Evaluation of WSN Attacks

In order to evaluate the proposed attack-modeling methodology, two examples of attacked WSNs are shown (Figs. 12.18 and 12.19). These networks and their attackers have been modeled in the virtual platform.

There two different types of node in both networks. The first type of node is the gateway or central node. It is responsible for communicating with the second type of node, the sensor node. The sensor node has a sensor to measure the environmental temperature. When the sensor reads this information, the node sends this information to the gateway. The gateway waits to receive information from sensors to generate a message with all the network information. This message

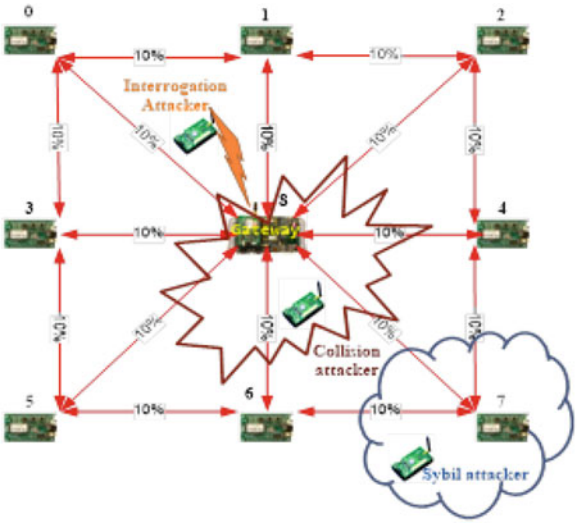


Fig. 12.18 Wireless mesh sensor network

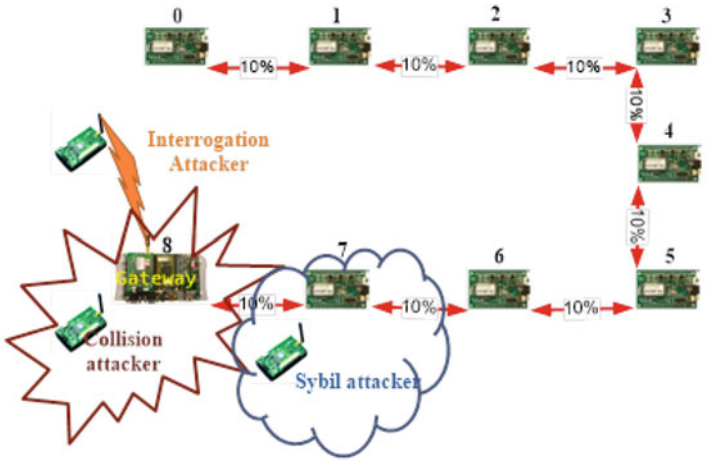


Fig. 12.19 Wireless linear sensor network

is sent through a GPRS module to a control center. When the nodes finish their functionality, they enter in a sleep mode to reduce the consumption and to increase the life of the node battery.

All the sensor nodes have the same hardware architecture and similar embedded software. In these scenarios, the platform model of the nodes includes an ARM 926 processor, a memory, a temperature sensor and an 802.15.4 transceiver. All these components are interconnected by a bus system. The embedded application software reads the sensor every 30 s and sends the information to a gateway node. The gateway analyzes the information and report to a GPRS network if detects an anomalous measurement.

In the first example (Fig. 12.18) the network has a mesh topology. Each node takes its measurement and sends it to the gateway. The percentages on the red lines represent the packet-loss probabilities of the wireless channel. If there is no red line between two nodes, it will be assumed that the packet- loss probability is 100 % (so, there is no direct connection).

The second network (Fig. 12.19) has a linear topology. Each node reads the sensor and sends the information to the next node of the network. Thus, the transmitted packet increases its size since it adds the information of each sensor while it passes through the whole network. The red lines in the Fig. 12.19 model the possible wireless communication channels and the packet-loss probability.

In both types of networks, three attacks have been modeled and simulated It is worth to mention, that the virtual simulator allows the simulation of all the attacks described in the previous sections.

Table 12.2 Simulation results

	Mesh network				Linear network			
	Gateway	Node (0–6)	Node 7	Total	Gateway	Node (0–6)	Node 7	Total
Attack-free (J)	3.62	0.72	0.72	9.32	0.57	1.6	1.6	13.4
Collision attack (%)	−28	+193	+193	+109	−9.5	0	+90	+9.4
Interrogation attack (%)	333	0	0	+90	+139	0	0	+65
Sybil attack (%)	0	0	+563	+68	0	0	+314	+21

12.5.3 Simulation Results

In this example, the first simulated attack is an Intelligent Collision attack on the gateways nodes with an effectiveness of the 60 %. The second attack is a Sybil attack on node 7 and the last one is an Interrogation attack on the gateway nodes. The objective of the simulations is to estimate which attacks are more dangerous (in terms of energy consumption) for each network.

Table 12.2 shows the estimated power consumption of each node in both networks. The power consumption of attack-free use case is used as reference. Therefore, the absolute power consumption values are shown for this use case. And the increase of power consumption is shown for the other cases.

As it can be observed in Table 12.2, the total power consumption of the Linear Sensor Network is 30 % higher than the Mesh Sensor Network. This is because of the increased workload of the sensor nodes since they must communicate with their neighbors. In contrast, the gateway power consumption can be lower (e.g. −9.5 %) because it receives fewer packets.

In the Collision attack case, the gateway reduces the power consumption because its workload is reduced since the attack decreases the number of packets received. However, it can be observed that the mesh network is more affected than the linear network in terms of power consumption when the total power consumption of the networks is compared. For example, the power consumption of the mesh network is increased by two while the linear network consumption is slightly affected by the Collision attack.

In the Sybil-attack case, the linear network also has a lower power consumption increase. These results do not mean that the linear network is more secure than the mesh network since the reduced power consumption is a consequence of a higher packet loss. In the Sybil attack, the mesh network loses 1/8 packets while the linear one loses all the packet information.

Thus, with these results a WSN designer can observe what topology and applications network are more vulnerable to some specific attacks. Therefore, it can design more secure WSN networks in non-secure environments.

12.5.4 Comparison Between Real Measurements and Virtual Platform Estimations

There have been real measurements performed in this evaluation. These measures were performed with the N6705b DC Power Analyzer from Agilent [2]. This device is a source/measure unit designed specifically for the task of battery drain analysis.

The working frequency of the processor of the sensors node and gateway is 120 Mhz. This is important since the power consumption is strictly related to this frequency. All the simulations that have been performed in this evaluation were done at this frequency. The real measurements are compared to the estimations provided by the virtual platform so the accuracy of the tool can be obtained.

Different scenarios have been simulated with the WSN deployment shown in Fig. 12.18. The goal is to simulate different features on the same environment and observe the WSN behaviour. After observing their results it is possible to decide about the hardware and software that the WSN nodes should include. The different scenarios are presented in the next paragraphs.

- **Scenario 1:** This scenario focuses on estimating the power consumption of the network normal operation mode. Each node performs some readings from their sensor, processes the sensor data and sends them to the gateway. They also receive messages from other nodes. This working mode does not encrypt the transmitted messages. The idea is to obtain an estimation of the power consumption of this network that can be used to compare against other scenarios.
- **Scenario 2:** This scenario is similar to the previous one but in this case the transmitted packets are encrypted by using an AES-256 scheme [18]. This is performed by all the nodes of the evaluation scenario. To accomplish this, the nodes use their hardware cryptography modules. Of course, it is also necessary to decrypt the received packets. There is also other type of packets (not related to data sensor). These packets are sent periodically and contain new cryptography keys. This is done to improve the security in the network. This will have also an impact in the power consumption of the whole network since more packets are transmitted compared to the scenario 1. The gateway sends new cryptography keys each 30 min in this simulation.
- **Scenario 3:** A common attack for wireless devices is simulated in this scenario. This attack is the Replay Attack and it consists in sending repeatedly packets to the node destination. These packets were previously captured by the RZUSB-STICK [3] so they can be identified correctly by the node victim. The idea behind this attack is to diminish the device battery by forcing the node to process useless packets.
- **Scenario 4:** The secure booting is a special feature of WSN nodes thus a special scenario is designed only for its evaluation. In this case, it is not necessary to simulate the whole network but only the node that is rebooted. The booting is composed of several steps that are also executed in the virtual platform. These steps are:

Table 12.3 Error of the virtual platform

	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
Error (%)	11	9	8	10	10	11

  - Read from the SD
  - Confirming authenticity and/or decrypting
  - Translation of the hex file
  - Saving the translated firmware to the FLASH
  - Application start

- **Scenario 5:** In the last scenario the whole network is modeled but the results are focused in one node since it tests a specials application. In this case, a normal firmware update is performed in one node. Since each packet has a maximum data size of 76 bytes, the whole firmware requires at least 1,752 packets.
- **Scenario 6:** This scenario focuses in simulating a partial firmware update. In this scenario the firmware size is 593 bytes. Same measurements as in the scenario 5 are performed so a comparison between both methods can be obtained.

The error in percentage of the Virtual Platform estimations is shown in Table 12.3. This error was calculated after comparing the estimations with the real measurements. In the cases that there were several measures, as in the scenario 4, the error average is shown.

It can be observed that the worst accuracy was obtained in scenarios 1 and 6 with an error of 11 %. The total error of the simulation tool varies from 8 to 11 % in the simulations performed in this evaluation.

References

1. Abdullah, M., Hua, G.W., Alsharabi, N.: Wireless sensor networks misdirection attacker challenges and solutions. In: IEEE International Conference on Information and Automation, 2008 (ICIA 2008), Hunan, pp. 369–373 (2008)
2. Agilent Technologies. (2014) http://www.home.agilent.com/en/pd-1842303-pn-N6705B/dc-power-analyzer-modular-600-w-4-slots?&cc=ES&lc=eng
3. Atmel Corporation. (2014) http://www.atmel.com/tools/rzusbstick.aspx
4. Becher, A., Benenson, Z., Dornseif, M.: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks. In: Proceedings of the Third international conference on Security in Pervasive Computing (SPC'06), pp. 104–118, Springer, York, UK (2006)
5. FreeRTOS. (2014) http://www.freertos.org/
6. Levis, P., Lee, N., Welsh, M., Culler, D.: Tossim: accurate and scalable simulation of entire tinyos applications. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03), Los Angeles, pp. 126–137. ACM, New York (2003). doi:10.1145/958491.958506. http://doi.acm.org/10.1145/958491.958506
7. Mekni, M., Moulin, B.: A survey on sensor webs simulation tools. In: Second International Conference on Sensor Technologies and Applications, 2008 (SENSORCOMM '08), Cap Esterel, pp. 574–579 (2008). doi:10.1109/SENSORCOMM.2008.13

8. Mohammadi, S., Jadidoleslamy, H.: A comparison of link layer attacks on wireless sensor networks. Int. J. Appl. Graph Theory Wirel. Ad Hoc Netw. Sens. Netw. **3**(1), 69–84 (2011)
9. Mohanty, P., Panigrahi, S., Sarma, N., Satapathy, S.S.: Security issues in wireless sensor network data gathering protocols: a survey. J. Theor. Appl. Inf. Technol. **13**(1), 14–27 (2010)
10. NS-2 The Network Simulator. (2014) http://www.isi.edu/nsnam/ns/
11. OMNeT++. (2014) http://www.omnetpp.org/
12. Pelechrinis, K., Iliofotou, M., Krishnamurthy, S.V.: Denial of service attacks in wireless networks: the case of Jammers. IEEE Commun. Surv. Tutor. **13**, 245–257 (2011). doi:10.1109/SURV.2011.041110.00022
13. Posadas, H., Castillo, J., Quijano, D., Fernandez, V., Villar, E., Martinez, M. SystemC platform modeling for behavioral simulation and performance estimation of embedded systems. In: Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation. IGI Global, 2010, pp. 219–243 (2014). doi:10.4018/978-1-60566-750-8.ch009
14. Posadas, H., Díaz, Á., Villar, E.: Sw Annotation Techniques and RTOS Modelling for Native Simulation of Heterogeneous Embedded Systems. INTECH Open Access Publisher (2012)
15. Reindl, P., Nygard, K., Du, X.: Defending malicious collision attacks in wireless sensor networks. In: 2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC), Hong Kong, pp. 771–776. IEEE (2010)
16. Tinyos. (2014) http://www.tinyos.net/
17. Torres, R., Valle, L., Domingo, M., Diez, M.: Cindoor: an engineering tool for planning and design of wireless systems in enclosed spaces. IEEE Antennas Propag. Mag. **41**(4), 11–22 (1999)
18. Wang, Y., Attebury, G., Ramamurthy, B.: A survey of security issues in wireless sensor networks. IEEE Commun. Surv. Tutor. **8**(2), 2–23 (2006). doi:10.1109/COMST.2006.315852

Chapter 13
Heap Management for Trusted Operating Environments

Iraklis Anagnostopoulos, Ioannis Koutras, Christos Andrikos, and Dimitrios Soudris

Abstract Dynamic memory managers are responsible for organizing the dynamically allocated data in memory and also servicing the application's memory requests (allocation/deallocation) at run-time. In today's trusted embedded systems, dynamic memory management is a mechanism implemented in order to interact with modern applications. However, the majority of these applications are not self secured. The combination of scripting languages, fast development and user centralized environments ends up with applications full of security flaws. In this chapter, we will present the Dynamic Memory Management (DMM) design space and all the orthogonal decision trees including heap protection actions. Also, we will present methods for securing memory allocators for modern embedded systems.

13.1 Introduction

High performance single-chip computing devices evolve from single- to multi- and even many-core architectures. In these cases, memories are preferably distributed for medium and large scale system sizes, because centralized approaches have become the bottleneck in performance, power and cost. Traditional memory optimization uses compile-time information and focuses on static allocation in respect to available memory hierarchy. For modern applications, where dynamicity is ascendant, this is no longer possible since there is a lot of memory unpredictability, which cannot be captured by source code analysis alone. Also, the increased dynamism in data storage leads to unexpected memory footprint variations unknown at design time.

Dynamic memory managers are responsible for organizing the dynamically allocated data in a part of memory called *heap* and also servicing the application's memory requests (allocation/deallocation) at run-time. In case of an allocation

I. Anagnostopoulos (✉) • I. Koutras • C. Andrikos • D. Soudris
School of Electrical and Computer Engineering, National Technical University of Athens, Athina, Greece
e-mail: iraklis@microlab.ntua.gr; joko@microlab.ntua.gr; candrikos@cslab.ntua.gr; dsoudris@microlab.ntua.gr

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_13

memory request for a new object, the dynamic memory manager returns to the application the pointer inside the heap, to the newly allocated object. In C/C++ programming language dynamic memory allocation is performed through the `malloc/new` operators. In the case of a deallocation memory request, the dynamic memory manager returns to the application either a true or a false value in respect to success of the process.

Dynamic memory management (DMM) is a critical component in modern embedded systems since the dynamic memory allocation often forms the main performance, and scalability bottleneck of modern complex applications. Also, it greatly affects the energy and memory consumption of the overall system.

Moreover, heap based attacks are an ongoing threat. As with many security vulnerabilities the exploitable glitch is discovered by the attacker. Also, bad memory management from the programmer may lead to various errors that could interfere with other processes and enable heap based attacks. Instead of relying on the programmers' memory managing skills, an approach to prevent these heap based attacks is to make the memory allocator safe. This basically means that the allocator tries to tolerate the inevitable memory errors and let them not interfere with anything but the processes that caused them.

13.2 Memory Segmentation

Process memory is divided to five main segments:

1. Text
2. Data
3. bss
4. Stack
5. Heap

Each segment represents a special portion of memory that is set aside for a certain purpose and has specific read/write permissions.

Text, also know as **code segment**: This segment contains the assembled machine language instructions that are to be executed. Due to the variety of the used function calls the execution of the instructions in this segment is non-linear. Also, read-only permission is allowed since its goal is to store instructions and not variables. Consequently, this memory segment has a fixed size. This read-only feature also allows the multiple execution of a program without problems. Any attempt to write to this segment of memory will cause the program to alert that something malicious happened and usually to be terminated.

Data and **bss segments** store global and static program variables and are placed consecutively after the text segment. On one hand the data segment is filled with the initialized global variables, strings and any other constants that are used by the process. On the other hand the **bss** stores all the uninitialized ones. Both segments

are writable as global variables' values usually alter during the execution. However because of the fact that the crowd of all global variables, strings and constants remains unchanged during the execution both segments have fixed size.

The **stack segment** is the first from the two segments whose size is variable, meaning it can become larger or shrink smaller depending on the program's needs. It is used as a temporary part in order to store context information between function calls giving the process the ability to remember caller's context after every internal context switch. When a process calls a function, that function contains its own set of variables and as aforementioned the code will be placed at a different memory location, the text segment, The stack segment is used to remember all the previous calls and values and the location that the instruction pointer register should return after the execution of the function. So, a new record has to be generated, containing the program counter's current value and the essential context before the switch ordered by function call mechanism. After its generation, the new stack record is placed (**push**) in the stack (in a *First In Last Out* way) segment whose size increases. When the called function ends the record is used to return to the hardware the state of the caller one. At this moment we got the pop functionality, decreasing the size of the stack.

Last but not least, the **heap segment** is used for the rest of the program variables, meaning all the dynamic allocated variables whose sizes are run-time estimated. It is the second segment type that has not fixed size as the stack segment does. All of the memory within the heap segment is managed by allocation and deallocation algorithms – mechanisms. These algorithms are used in order to reserve memory from the system, so as to serve the requests of the program and also release any already used memory that it is not needed anymore.

13.3 Heap Attacks

In a heap attack the attacker targets the heap memory of a process. As aforementioned, the heap contains all the memory the current process has allocated and if the attack succeeds the attacker could compromise the system, or simply just cause a crash. There two conditions that are inevitable prior to a heap attack:

- First of all there has to be a memory management error.
- Secondly the memory allocator has to be exploitable.

13.3.1 Memory Management Errors

It is very hard to prove that an application is safe from memory errors. This happens because memory management errors come from the programmer. It is very common programmers to use memory in a more care less way creating "holes" that can

be used for an attack. Especially, unsafe programming languages (such as C and C++) do not provide memory error checking. There are five common memory management errors:

- **Heap overflow and underflow**: Overflow happens when the allocated memory is too small to store the object and the data overexceed. On the opposite, underflow happens when a read or write performed before the allocated memory. Overflows happen way more often than underflows.
- **Dangling pointers**: Dangling pointer is called the pointer that points to an already freed object. These pointers are also known as use-after-free pointers.
- **Double free**: If the programmer tries to free the same object twice, double free memory management error occurs. Freeing the same memory chunk more than once can corrupt memory manager data structures in a manner that is not immediately apparent.
- **Invalid free**: This error happens if the code tries to free an object that no memory has been allocated beforehand.
- **Unitialized reads**: In unsafe programming languages (such as C and C++) allocating a memory space does not mean that it is also initialized. So, incorrect usage of unitialized memory leads to memory management error and unexpected behavior.

13.3.2 Exploitable Allocator

Assuming that an attacker has found out a memory management error and knows the execution path to the error, the next point of interest is the memory allocator. Almost all widely used memory managers are predictable, meaning that we can guess and predict how the allocated memory is organized in the heap. Thus, allocator exploitation is achieved through driving the allocator to a predictable state. For example, Sotirov describes a sophisticated technique called Heap Feng Shui that allows attacks on browsers running JavaScript to ensure predictable heap behavior [14]. The most common attacks are:

- Heap overflow attacks
- Heap spraying attacks
- Dangling pointer attacks
- Off by one error

An attack mostly aims heap metadata in order to change the allocator's behavior through heap's metadata illegal modification. Attacking the application's data, offer the attacker an easily exploitable environment though a special behavior of a running process that the programmer has not forecasted.

13.3.2.1 Heap Overflow Attacks

Heap overflow attack is the most common attack. As aforementioned, the heap implementation divides the heap into manageable chunks and tracks which heaps are free and in use. Each chunk contains a header structure and free space (the buffer in which data is placed). The header structure contains information about the size of the chunk and the size of the preceding chunk (if the preceding chunk is allocated). The main goal of this specific technique is to overwrite essential heap informations in the allocated chunks and in that way alert allocator's functionalities. The most common scenario is to focus on metadata aiming to pointer overwriting resulting to forward and backward consolidation attacks.

13.3.2.2 Heap Spraying Attacks

Heap spraying attacks [9] are used to make exploitation of other vulnerabilities simpler. On many systems, the heap lies within a restricted address space and during a heap spaying attack, the application code can often be made to read an address from an arbitrary location in memory. The heap has been "sprayed" with shellcode and the attacker tries to force the application to read an address from the sprayed heap. However, to successfully exploit a heap spray, the attacker must guess the address contained within some (large) set of attacker-allocated objects. In modern systems, guessing the location of heap-allocated shellcode or the address of a specific function can be difficult due to ASLR [11].

13.3.2.3 Dangling Pointers

Dangling pointers are pointers to memory locations that are no longer allocated. In most cases this will lead to a program crash but in some cases it could lead to a double free vulnerability. Such a double free vulnerability could be used by the attacker in order to modify or alert the behavior of the allocators resulting in a code injection attack [7, 18]. However, dangling pointers is prevented in most modern memory allocators because these allocators check if a chunk is freed twice in a row.

13.3.2.4 Off by One Error

An off by one error is a special case of the buffer overflow. When an off by one occurs, the adjacent memory location is overwritten by exactly one byte. In some cases these errors can also be exploitable by an attacker. A more generally exploitable version of the off by one error for memory allocators is an off by five.

13.4 Dynamic Memory Management Design Space for Trusted Operating Environments

In order to design or customize an efficient dynamic memory manager a number of decisions and strategies have to been explored. Each decision forms a different implementation choice. Different combination of decisions delivers different dynamic memory managers. Thus, the specification of all the possible decisions and strategies concerning dynamic memory allocation has to be defined in order to be able to explore various alternatives.

The enumeration of these decisions defines the complete design space of dynamic memory management. All of them should affect as less as possible the other ones, i.e. be as orthogonal as possible. Thus, this set of possible decisions must cover exhaustively any kind of potentially profitable (In the Pareto trade-off sense, where more than one metric is considered and a solution can be very better in general, but not for one axis) dynamic memory scheme that exists currently in the literature. Furthermore, we propose to use taxonomy of decisions based on orthogonal trees at two different abstraction levels.

Before presenting the design space exploration for Dynamic Memory Management there is need to present some of the implemented techniques for securing heap and the necessity for building secure memory allocators in a modular way. Authors in [5] propose a non-executable heap. Non-executable pages are supported in virtual memory hardware and in recent versions of various Operating Systems, such as Microsoft Windows XP Service Pack2 [1], Linux and Open BSD. However, some embedded systems may not be able to even host an operating system due to limited storage but this cannot exclude them from being used in trusted computing platforms leaving heap management vulnerable to attacks. Authors in [10] present program shepherding, a technique through which policies regarding control flow transfer can be specified and enforced. However, both non-executable heap and program shepherding try to prevent the last stage of an attack. Attacks that do not rely on control flow modifications cannot be prevented as presented in [4]. PointGuard [6] stores encrypted pointers in the main memory and recently it has been implemented as a hardware approach for stack protection [16]. The compiler or binary rewriter is required to identify type information, and apply encryption/decryption on pointer accesses. However, without sacrificing language compatibility, accurately identifying pointer accesses is difficult in C programs due to lack of type information. Transparent Runtime Randomization(TRR) [17] and Address Space Layout Randomization (ASLR) [15] are software techniques that randomize the starting address of various segments (heap, stack, BSS, etc.) and dynamic library codes when a program is loaded [17]. Although TRR and ASLR increase the difficulty of attacks that involve more than one segment, they cannot protect against attacks that are solely carried out within the heap segment.

Based on the analysis presented in [2] a taxonomy of the available decisions through a set of orthogonal decision trees has been presented. According to their taxonomy, we extend the DMM design space in order to efficiently model and capture

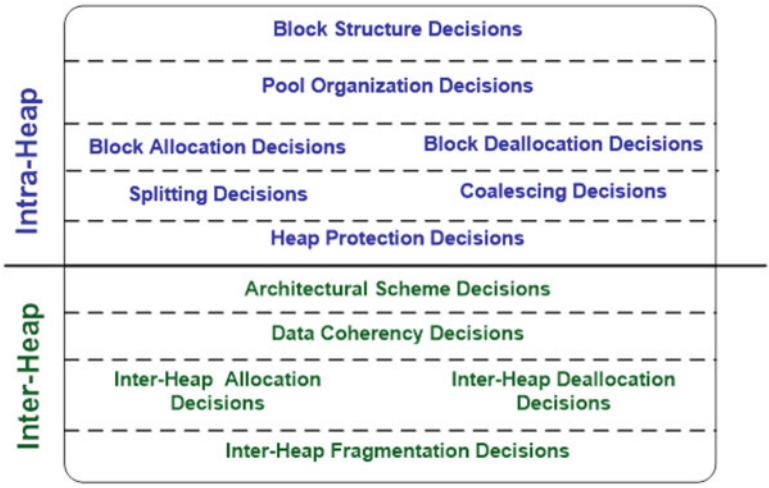


Fig. 13.1 DMM design space

trusted computing decisions for securing heap management. The extended design space remains platform independent and applicable to any MPSoC platform after a platform dependent refinement. The design space is defined through platform-independent Decision Trees (DTs) that capture both the inter- and intra- heap level decisions for composing secure software customized dynamic memory allocators. An overview of the DMM design space is presented in Fig. 13.1.

13.4.1 Intra-thread Design Space

- **Block Structure Decisions** which handle the way block data structures, are created and used by the dynamic memory managers to satisfy the memory requests inside each heap.
- **Pool Organization Decisions** which deal with the number of pools present in each heap and the reasons why they are created.
- **Block Allocation Decisions** which deal with the actual actions required in intra-thread level dynamic memory management to satisfy the memory requests and couple them within a memory block.
- **Block Deallocation Decisions** that address the need of freeing the memory block when it is no longer needed (and explicitly freed by the application running).
- **Splitting/Coalescing Blocks category** that refers to the actions executed by the dynamic memory managers to ensure a low percentage of memory internal fragmentation of each heap, namely splitting one larger block into two smaller ones.
- **Heap Protection Decisions** refer to the actions required in order to secure heap against heap attacks. Heap attacks, based on memory management errors,

can lead to application crashing or even worse to the execution of malicious code. Some of the most common heap attacks are heap overflows, dangling pointer, double frees and invalid frees.

13.4.2 Inter-thread Design Space

- **Architectural Scheme Decisions** which determine the way the dynamic memory allocator organizes and architects its heaps in order to exploit the available thread-level parallelism into memory management.
- **Data Coherency Decisions** which deal with the existence or not and the structure of the synchronization mechanisms in order to ensure the data coherency in each heap. The Data Coherency Decisions are modeled through a single orthogonal tree. Since multiple-threads are running in parallel multiple memory requests can be rose concurrently from different which may corrupt the data and subsequently the functionality of the dynamic memory manager and the application, if special care of the coherency mechanisms is ignored.
- **Inter-Thread Allocation Decisions** which manage the way in which threads allocate memory in the inter-thread level. Allocation in this level is strongly connected with decisions which consider both the thread grouping in order to share a heap and the thread to heap mapping. Allocation decisions of finer granularity i.e. fit policies etc. are included into the intra-thread design space. At the inter-thread level, allocation decisions reduce to the mapping of threads to heaps. Specifically, this type of decision determines which of the available heaps in the dynamic memory manager will serve the memory allocation request according to the thread's ID.
- **Inter-Thread Deallocation Decisions** which include trees concerning the ownership aware, deallocation of each memory block and placement decisions for the deallocated blocks.
- **Inter-Thread Fragmentation Decisions** manages the potential memory blow-up of the multi-threaded application and considers decisions in order to reduce or bound the worst memory blow-up.

13.4.3 Interdependences Between Data Management Techniques

After the definition of the orthogonal trees for dynamic memory management, in this section we identify their possible interdependencies. Evidently, although the decision trees are orthogonal, the selection of certain leaves in some trees affects heavily the coherent decisions of the other ones. Thus, they possess interdependencies to take into account when we analyze a certain dynamic memory manager.

These inter-dependencies can be classified in two main groups. First, the inter-dependencies caused by certain leaves, trees or categories, which disable

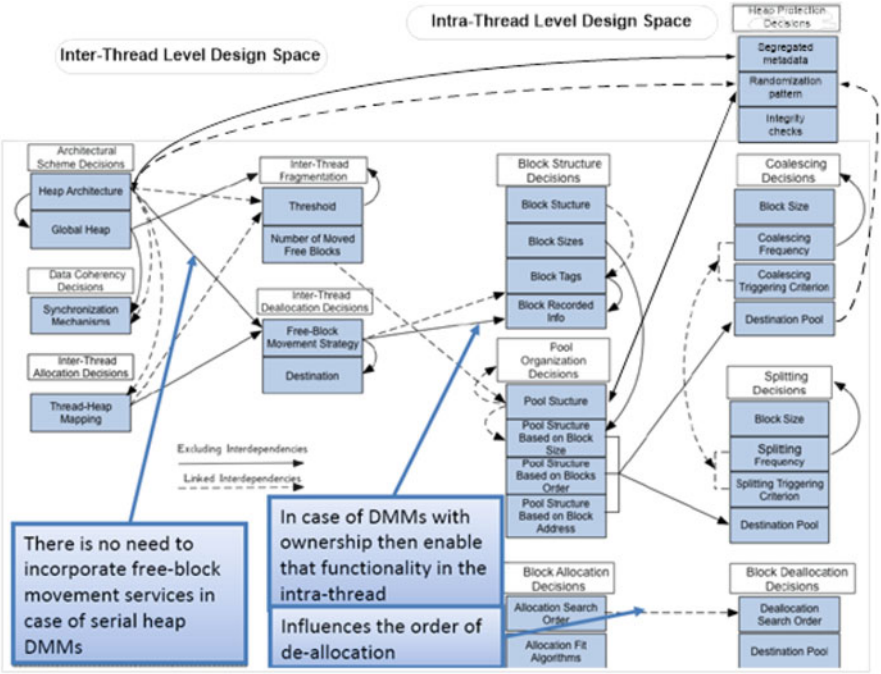


Fig. 13.2 Interdependences between data management techniques

the use of other trees or categories. We call this type of interdependencies as **excluding interdependencies**. They are drawn in Fig. 13.2 as full arrows. Also, there exist inter dependencies that affect other trees or categories due to their linked purposes (e.g. coalescing/splitting etc.). These interdependencies are called **linked interdependencies**and they shown in Fig. 13.2 as dashed arrows.

It is important to mention also that, in Fig. 13.2, two different kinds of endings for the arrows are present. First, the double ending arrows mean mutual influence that cannot allow identifying clearly an order (which one should be characterized first) in the overall cost produced by these categories without the addition of more information to the exploration (e.g. specific function cost based on the number of accesses, memory size or control flow, definition of the final architecture, etc.) Second, the single ending arrows indicate that the side without arrow ending affects the other one and must be selected first.

13.5 Heap Attack Prevention Techniques

As aforementioned, most memory allocators ignore security issues. Instead they focus on maximizing performance and limiting fragmentation and waste. DieHard [3] is a bitmap based allocator and was originally designed to be resistant against

memory errors. This resistance was achieved through randomizing both memory allocation and the reuse of freed objects. These properties greatly improved reliability in two different ways. One of them is that randomization minimized the possibility of two adjusting blocks to be one next to the other and thus overflow attack is prevented. The second one is that the randomization makes the allocator unpredictable. DieHard provides greater security guarantees than other general-purpose allocators. However, DieHard was designed to increase reliability against memory errors rather than for increased security and so DieHard enables the program to continue running after experiencing memory errors. There is an adaptive version of DieHard, called DieHarder [11], that manages memory through so called miniheaps. Each of these miniheaps holds objects of exactly one size and each miniheap size is adaptively adjusted according to a predefined ratio between used and allocated space. Similarly to OpenBSD, DieHarder randomly allocates individual pages from a large section of address space carving them up into size-segregated chunks. In [12] authors present a technique that protects the heap management information and allows for run-time detection of heap-based overflows. Each chunk is protected by a randomly-seeded checksum over its metadata fields. So, whenver an action is required for that chunk, the ckecksum is checked in order to ensure that no over- or underflow has occured. Also, each access of a list pointer is protected by a check to insure that the integrity of the pointers has not been violated.

However, all the aforementioned techniques, require dependency on operating systems and in many times there is big memory overhead, especially in replication and heap randomization techniques. Heap protection in embedded systems require faster and less aggressive, in terms of memory, solutions. The presented protection mechanisms especially designed for trusted embedded systems are canaries, encrypted pointers and encrypted lists. All the supporting metadata of the metadata protection mechanisms use a *miniheap* scheme to be stored to. It is actually implemented in a mini bibop-based allocator in the original main allocator that stores all the appropriate information. The specific miniheap is in the same address space that the concluder one, but the way that it grows and the actual meaning of its contents are generally encrypted. Miniheap is actual a minimal allocator inside the application's allocator itself. Only the running process is able to know the way to find the contents it needs for running appropriately. Even the main allocator does not "see" the address of the elements that the miniheap keeps. In the miniheap, we save the keys for all allocated blocks and whenever there is an action regarding the blocks, we verify the integrity of this action. For that reason, we have created two structures for keeping the keys:

- Single Linked Lists (SLL) [13]
- Adelson-Velskii and Landis' (AVL) tree [8]

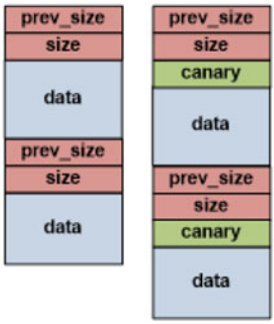


Fig. 13.3 Canaries mechanism

13.5.1 Canaries

The canary mechanism contains a checksum of the chunk header seeded with a random value Fig. 13.3.

When a chunk is returned to the heap management through a call to free, the chunk's canary is checked against the checksum calculation performed when the chunk was released to the application. If the stored value does not match the current calculation, a corruption of the management information is assumed. At this point, an alert is raised, and the process is aborted. Otherwise, normal processing continues; the chunk is inserted into a bin and coalesced with bordering free chunks as necessary. Any free list manipulations which take place during this process are prefaced with a check of the involved chunks' canary values. After the deallocated chunk has been inserted into the free list, its canary is updated with a checksum covering its memory location, size fields and list pointers. This mechanisms is effective against heap overflow attacks.

13.5.2 Encrypted Pointers

Concerning security, the dynamic memory manager can encrypt the pointers in the metadata field so as not to reveal the address of the next block. The allocator decrypts the pointers only when it is necessary to know the real addresses before dereferencing Fig. 13.4.

This mechanisms is mostly effective against double free heap attacks and it also prevents in some cases heap overflow attacks.

13.5.3 Encrypted Lists

Dynamic memory management is based on "chunks", memory blocks that consist of application usable regions and additional in-band management information. A header is attached to each allocated object and contains its size and the size

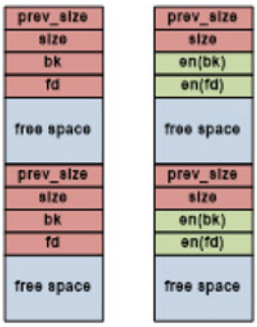


Fig. 13.4 Encrypted pointers

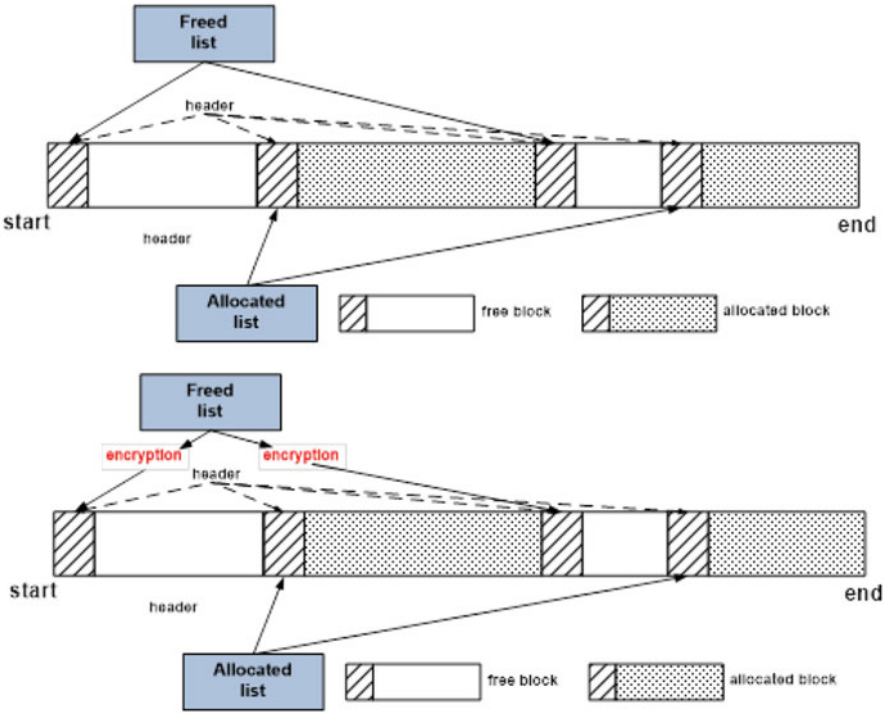


Fig. 13.5 Encrypted lists

of its previous object connected them in a linked-list style. The allocator uses a list structure (single or double linked) for organizing all this kind of information. In this level of organization we have developed an encryption mechanism in order to prevent attacks in consecutive nodes of the list. Each node can be considered as a memory page containing allocated or freed blocks. The list is considered as a higher-level organization of blocks-pages returned by the system. Encryption in this level of organization can achieve great protection against heap attacks to allocator's metadata and higher structures (Fig. 13.5).

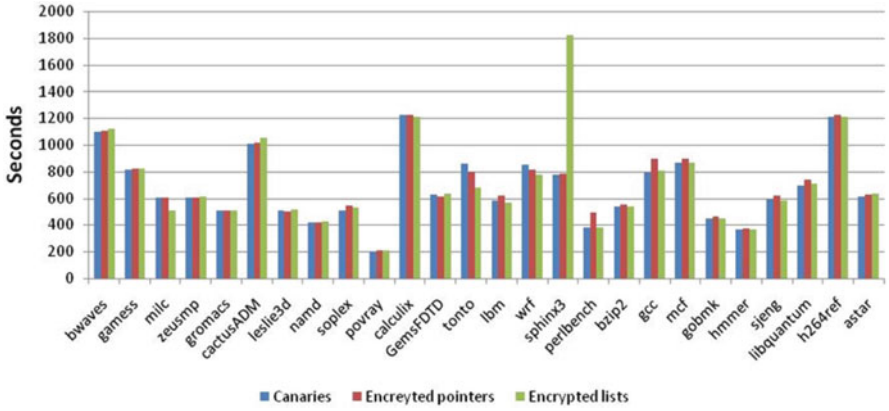


Fig. 13.6 Comparison of static implementations

Figure 13.6 presents the comparison of the static implementation of canaries, encrypted pointers and encrypted lists. We can see that in most cases, all implementations have the same performance in terms of seconds and thus the combination of them can be an effective way to prevent and defend the heap.

References

1. Andersen, S., Abella, V.: Data execution prevention. Changes to functionality in microsoft windows xp service pack 2, part 3: memory protection technologies (2004). http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/sp2mempr.mspx
2. Atienza, D., Mendias, J.M., Mamagkakis, S., Soudris, D., Catthoor, F.: Systematic dynamic memory management design methodology for reduced memory footprint. ACM Trans. Des. Autom. Electron. Syst. (TODAES) **11**(2), 465–489 (2006)
3. Berger, E.D., Zorn, B.G.: Diehard: probabilistic memory safety for unsafe languages. In: ACM SIGPLAN Notices, vol. 41, pp. 158–168. ACM, New York, NY, USA (2006)
4. Chen, S., Xu, J., Sezer, E.C., Gauriar, P., Iyer, R.K.: Non-control-data attacks are realistic threats. In: Usenix Security, Baltimore, vol. 5 (2005)
5. Conover, M.: w00w00 on heap overflows (1999). http://www.w00w00.org/files/articles/heaptut.txt
6. Cowan, C., Beattie, S., Johansen, J., Wagle, P.: Pointguard tm: protecting pointers from buffer overflow vulnerabilities. In: Proceedings of the 12th Conference on USENIX Security Symposium, Washington, DC, vol. 12, pp. 91–104 (2003)
7. Dobrovitski, I.: Exploit for cvs double free() for linux pserver (2003). http://seclists.org/lists/bugtraq/2003/Feb/0042.html/
8. eTutorials: Avl trees. http://en.wikipedia.org/wiki/AVL_tree
9. Gonchigar, S.: Ani vulnerability: history repeats, SANS Institute (2007)
10. Kiriansky, V., Bruening, D., Amarasinghe, S.P.: Secure execution via program shepherding. In: USENIX Security Symposium, San Francisco, vol. 92 (2002)
11. Novark, G., Berger, E.D.: Dieharder: securing the heap. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, pp. 573–584. ACM (2010)

12. Robertson, W.K., Kruegel, C., Mutz, D., Valeur, F.: Run-time detection of heap-based overflows. In: LISA, San Diego, vol. 3, pp. 51–60 (2003)
13. Single list. https://en.wikipedia.org/wiki/Linked_list#Singly_linked_list
14. Sotirov, A.: Heap feng shui in javascript. In: Black Hat Europe, Amsterdam (2007)
15. Team, P.: Pax address space layout randomization (aslr) (2003). http://pax.grsecurity.net/docs/aslr.txt
16. Tuck, N., Calder, B., Varghese, G.: Hardware and binary modification support for code pointer protection from buffer overflow. In: 37th International Symposium on Microarchitecture, 2004 (MICRO-37 2004), Portland, pp. 209–220. IEEE (2004)
17. Xu, J., Kalbarczyk, Z., Iyer, R.K.: Transparent runtime randomization for security. In: Proceedings of the 22nd International Symposium on Reliable Distributed Systems, 2003, Florence, pp. 260–269. IEEE (2003)
18. Younan, Y., Tenebras, V., Younan, D.Y., Vermeir, D.: An overview of common programming security vulnerabilities and possible solutions, Katholieke Universiteit, Leuven (2003)

Chapter 14
IP-XACT Extensions for Cryptographic IP

Emmanuel Vaumorin, Bernard Kasser, Sylvain Duvillard, and Albert Martinez

Abstract This chapter aims at explaining the implementation of security attributes in the IP-XACT (IEEE1685) standard. First, the state of the art usage of IP-XACT in the current design flow of industry will be introduced. In a second part, considering the stakes of security domain in System-on-Chip, we will demonstrate how IP-XACT improves the design flow. Finally we will detail the extensions of the IP-XACT standard that allow the coverage of security aspects in System-on-Chip design.

14.1 Introduction

Many modern systems employ security and cryptographic IPs in order to offer secure services and prevent exploitation attacks. These IPs are integrated using the System-on-Chip technology offering better level of security, performance and cost reduction.

The cryptographic IP that we will focus in this chapter has been designed by STMicroelectronics [2] and it is a hardware accelerator that provides cryptographic services to the system. The system may enable the usage of this IP's services to several users. This IP is an intelligent DMA: from an interpreted sequence code, the bloc sends instructions through crypto channels to crypto IPs. Figure 14.1 shows the architecture of a system using the cryptographic DMA.

It is possible to dispatch in parallel the instructions on cryptographic channels. Channels will charge the keys on specified locations. In a debug purpose, the registers are accessible through a slave interface, but in a delivery manner, this mode

E. Vaumorin (✉) • S. Duvillard
Magillem, 4 Rue de la Pierre Levée, 75011 Paris, France
e-mail: vaumorin@magillem.com

B. Kasser • A. Martinez
STMicroelectronics, Rousset, France
e-mail: bernard.kasser@st.com; albert.martinez@st.com

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5_14

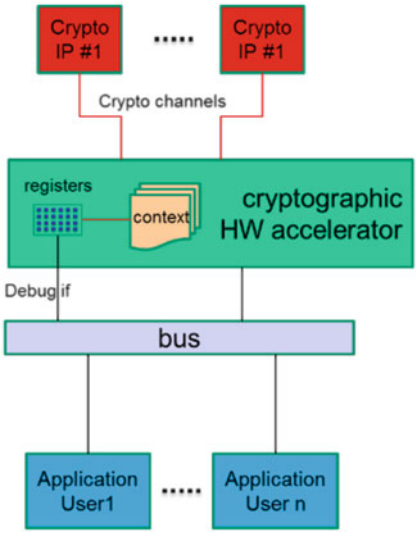


Fig. 14.1 Architecture of a system using the cryptographic DMA

shall not be possible. The IP does not have an action on the channels themselves. It is possible to multiplex tasks at a coarse grain. Thus, it may be possible to share a same cryptographic IP through several users but this implies to guaranty tightness between each context, may they be saved or not.

In respect with security requirements, a driver provided for this IP must be trusted and thus must be implemented carefully; for instance, the access to registers that are tagged as "security related" must be restricted; indeed, these registers give access to cryptographic channels and thus potentially this will let the application layer the capability to hack the IP and get confidential information. We will see in further part how they are tagged in an excel sheet and how IP-XACT [1] extensions may be used to enhance the respect of this security rule. Nevertheless, the cryptographic channels may be set accessible through the slave interface in the debug mode.

In respect with security requirements, a driver provided for this IP must be trusted and thus must be implemented carefully; for instance, the access to registers that are tagged as "security related" must be restricted; indeed, these registers give access to cryptographic channels and thus potentially this will let the application layer the capability to hack the IP and get confidential information. We will see in further part how they are tagged in an excel sheet and how IP-XACT extensions may be used to enhance the respect of this security rule. Nevertheless, the cryptographic channels may be set accessible through the slave interface in the debug mode.

Thus we will try to strengthen the level of information that is contained in the IP package by using extended IP-XACT and reflect this information at the driver level, even at the application level.

14.2 Presentation of the Approach

Our goal is to apply a new approach already defined for the co-design of low level functional sequences: aligning specifications (e.g. excel sheet), hardware description (better using IP-XACT) and code source (using sequences description model).

A typical embedded development workflow produces a very large number of artifacts: specifications, IP descriptions, verification code, driver code, BOM, etc. Maintaining consistency between two iterations of a platform (or of a single IP) is time-consuming and sometimes even impossible. Tracing requirements and evaluating impact of changes on artifacts is a transversal process of its own.

The most current issues are due to evolution of the previous material: bit fields and registers moved around, deprecated registers and bit fields (worst case being silent fails), working binaries with no clear specification or code attached, etc.

Partial solution already exist but have limitation: versioning system in word documents, filename-based versioning of PDF document, SCMs (ClearCase, SVN, Git) for structured platform descriptions and actual code, XLS/CVS files for keeping track of everything. But the general shortcomings are the followings: there is no actual integration between file formats/versions and no effective traceability between artifacts, no environment for handling efficiently a design flow's life-cycle (from specification down to implementation, tracing each requirement down at the IP and driver code level, handling the impact of modifications made on one element over the whole set of artifacts).

Figure 14.2 highlights the possible inconsistencies and potential misunderstanding when SW engineers have to develop an application exploiting hardware resources: the hardware components are delivered with specifications documents and technical data-sheets, containing explicit and implicit information on how to use it, configure it, program it. Our goal is to use a standardized way to format these documents. The risk with misunderstood specification is to develop a driver that is not fully compliant with hardware specifications: for instance with the ST cryptographic IP, if the debug mode is not removed from the functions provided by the released driver, the security will be impacted, as there will be a way for the application to access to a forbidden service and get the cryptographic keys. In a more accurate approach, we will see that some registers of the IP are defined as "security related" because they are storing ciphering related data; we will see that these "sensible" registers are not explicitly annotated, but just colored in blue in the excel sheet; this may be risky if the software engineer don't take it into account, and even if we can trust the work of software engineer, it does not dispense to apply a checking step by using more explicit specification based on a standard in order to strengthen the heterogeneous value chain of actors.

Figure 14.3 illustrate the several step that we have experimented in order to prove the feasibility of the deployment of our approach in an industrial context. The left zone is dedicated to the hardware IP design phase; we will show how it is possible to ameliorate some packaging steps, relaying on the IP-XACT standard and propose

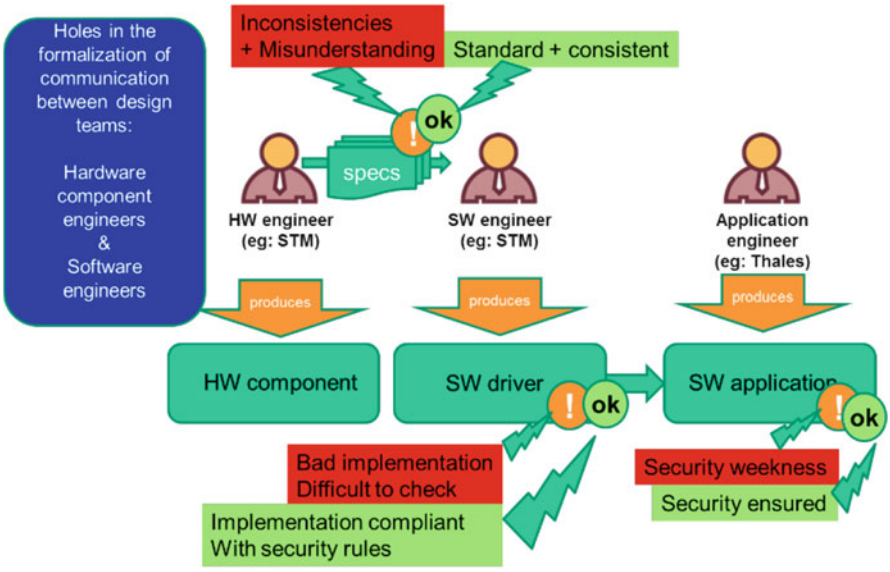


Fig. 14.2 Advantages of our approach in the heterogeneous value chain

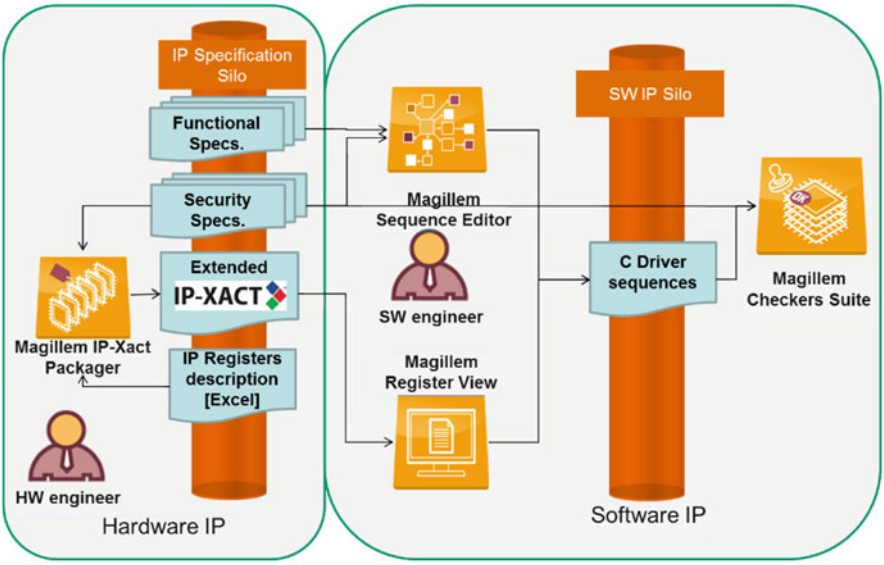


Fig. 14.3 Experimentations based on an industrial use case

some extensions. The right zone is dedicated to the software IP development; we will show how the previous work done at the hardware level will be exploited in order to automate some design steps and bring some checking capabilities and reduce probability of security weakness at application level.

We will see in the Sect. 14.3 part how we will apply this approach to the use case: registers zones will be tagged as "restricted access" using the IP-XACT extended for security; then the implementation of the driver will be checked in regards with this specification in order to validate it or report an error.

Another application will be to declare a register zone as a container for cryptographic key; we will offer the capability to validate the fact that a driver is well deleting this critical value at the end of specific sequence.

14.3 Experimentations

14.3.1 General Objectives of the Demonstrator

The hardware IP provider knows where is the weakness of its IP and thus must provide to the system integrator the list of possible attacks (well known or specific) and corresponding security rules and eventually security artifact that must be deployed in the system (hardware or software).

The IP documentation is spread among several files and IP-XACT is a mean to centralize in a standardized way a big amount of information. We want to show that the IP integration can be ameliorated for security purpose, using structured information that is available in the implementation flow. This will mainly relay on IP-XACT files including extensions for security.

14.3.2 IP-XACT Packaging with Extensions

The first step of the demonstrator is to package the cryptographic IP in IP-XACT.

14.3.2.1 Step 1: Import IP Registers Description

An Excel sheet is containing the complete description of the register map as depicted in Figs. 14.4 and 14.5. In the industrial flow, this Excel sheet is used to generate an XML file, thanks to a macro; this file is used to generate the data-sheet in frame maker (using notes) and to generate a verification code in C.

We have configured the importer in order to specify the format of the excel file used as input and get the IP-XACT equivalent file, containing all the registers description as depicted in Fig. 14.6.

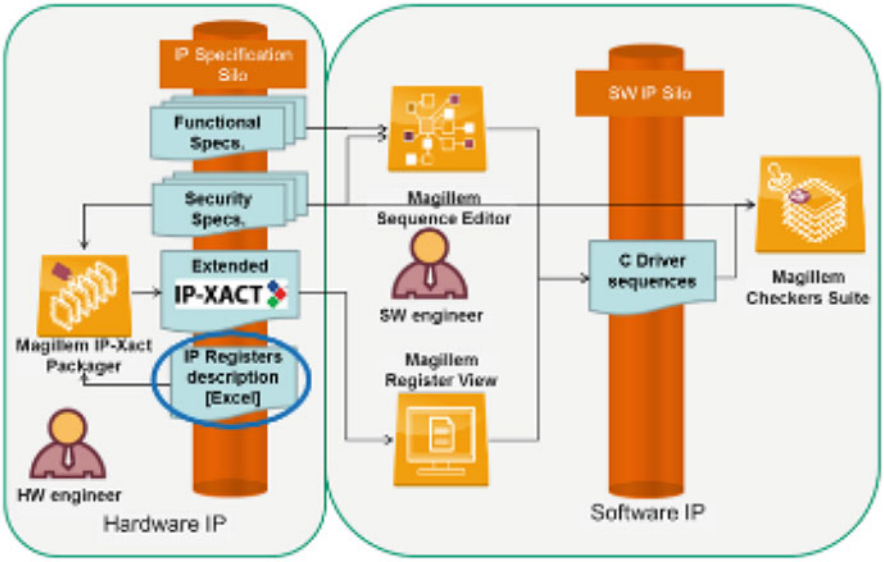


Fig. 14.4 Step 1 – IP registers description

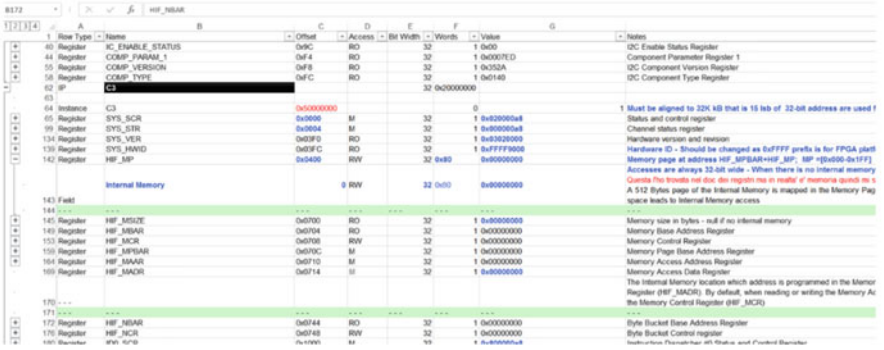


Fig. 14.5 Excel sheet containing the registers description of the IP

Registers depicted in blue color and bold police are related to crypto channels access and shall be considered with a particular attention; these registers may be made available from a slave interface for debug purpose but must not be accessed in a production mode, otherwise security is not preserved.

14.3.2.2 Step 2: Extend IP-XACT with Security Information

We have determined three extensions that are needed in IP-XACT to manage security information; these extensions are referenced (PSS-1, PSS-2, PSS-3) (Fig. 14.7).

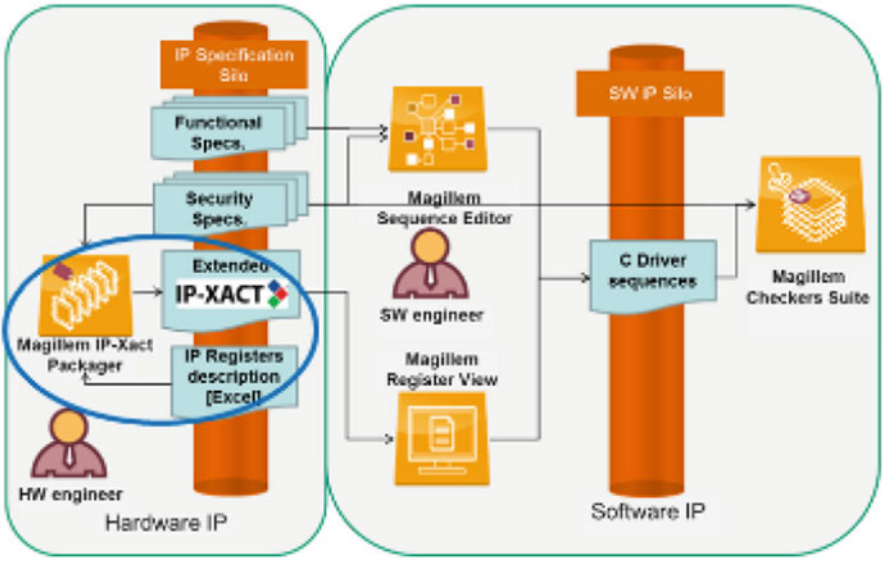


Fig. 14.6 Importing registers description in IP-XACT

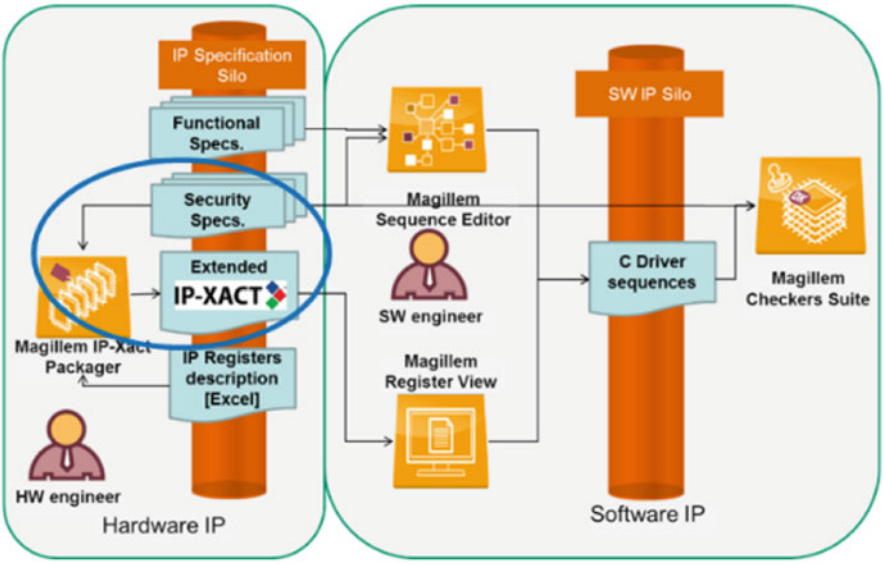


Fig. 14.7 Step 2 – extend IP-XACT for security relative information

We will use a parameter in IP-XACT register description to determine a security relative information. In the excel sheet which describes the registers of the IP, we can find information related to security (Figs. 14.8 and 14.9):

Fig. 14.8 Registers characterized by color

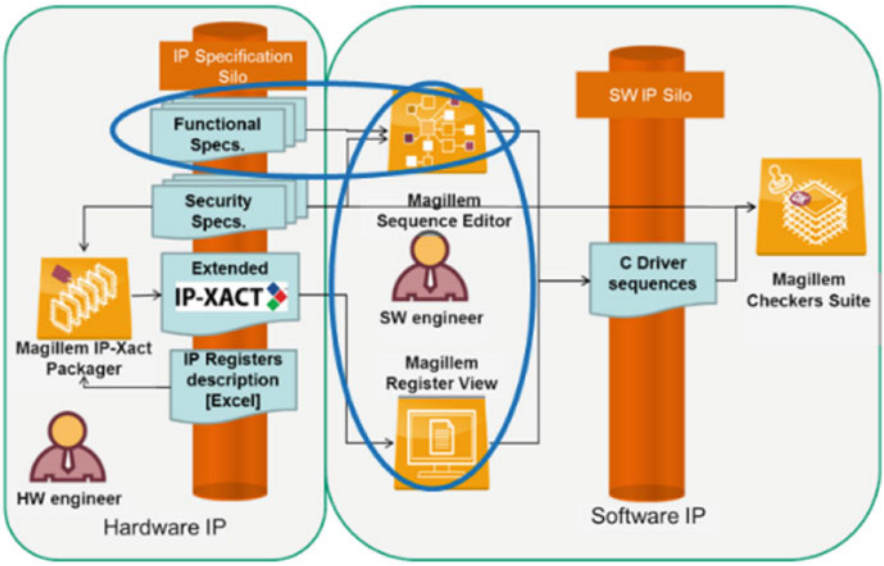


Fig. 14.9 The driver is described in sequences using a link with the IP-XACT register definition

1. Registers depicted in black color are related to the infrastructure of the IP and are normally used by the register to configure the IP (write access) and get useful information (read access).
2. Registers depicted in blue color and bold police are related to crypto channels access and shall be considered with a particular attention; these registers may be made available from a slave interface for debug purpose but must not be accessed in a production mode, otherwise security is not preserved.

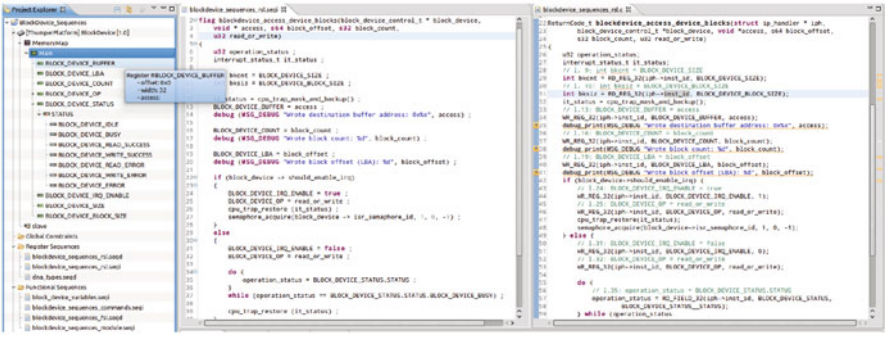


Fig. 14.10 Example of a driver sequences description using IP-XACT references

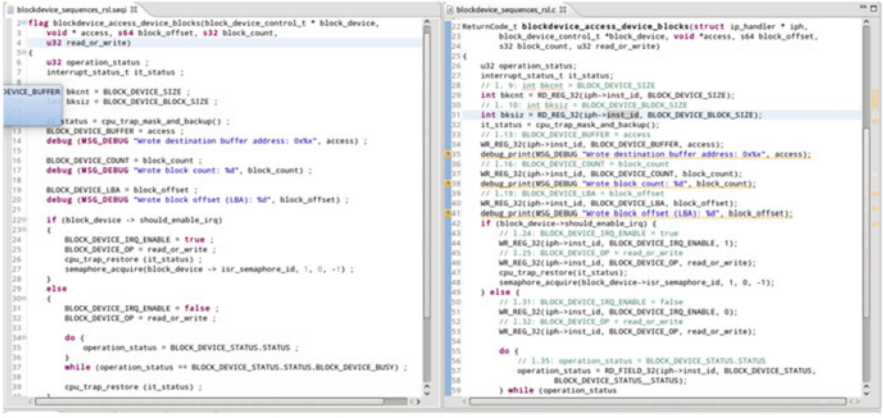


Fig. 14.11 Generated C code

14.3.2.3 Step 3: Help in the Implementation of Software Driver

A tool is used to describe the structure of the IP driver as well identified sequences. Figure 14.10 presents an example of a structured description of sequences (right side) using direct references with the register description in IP-XACT (right side).

The tool is able to define a register that contains a state and may be used to define valid and invalid values in another register in function of this state. The C code is then generated from the sequence description. Thus, Fig. 14.11 shows a C code that has been generated for the sequences description.

Conclusions
In this chapter we presented that the exchange of information between hardware provider and software developers is crucial, especially when security is at the heart of the preoccupations. A first answer is to use the standard IEEE 1685 IP-XACT that offers an accurate way to describe the detailed hardware architecture that will be accessed by software layers. Nevertheless, security related specifications must also be taken into account in order to avoid implicit information, and to do so we have proposed extensions of IP-XACT. The presented approach allows to understand how a methodology may be applied at several steps of the flow using specific tools in order to strengthen the delivered products in regards with potential security hacks.

References

1. Ip-xact. http://en.wikipedia.org/wiki/IP-XACT
2. Stmicroelectronics. http://www.st.com

Index

B. Candaele et al. (eds.), *Trusted Computing for Embedded Systems*,
DOI 10.1007/978-3-319-09420-5