Benoit Badrignans · Jean Luc Danger
Viktor Fischer · Guy Gogniat
Lionel Torres *Editors*

Security Trends for FPGAS

From Secured to Secure Reconfigurable Systems

Springer

Security Trends for FPGAS

Benoit Badrignans • Jean Luc Danger •
Viktor Fischer • Guy Gogniat • Lionel Torres
Editors

Security Trends for FPGAS

From Secured to Secure Reconfigurable Systems

Springer

Editors
Benoit Badrignans
Netheos
Montpellier
France
b.badrignans@netheos.net

Jean Luc Danger
Telecom Paris-Tech
Paris
France
danger@enst.fr

Viktor Fischer
Hubert Curien Laboratory—UMR CNRS 5516
Jean Monnet University
Saint-Etienne
France
fischer@univ-st-etienne.fr

Guy Gogniat
Lab-STICC—UMR CNRS 3192
Bretagne-Sud University
Lorient
France
guy.gogniat@univ-ubs.fr

Lionel Torres
LIRMM—UMR CNRS 5506
University of Montpellier 2
Montpellier
France
lionel.torres@lirmm.fr

ISBN 978-94-007-1337-6 e-ISBN 978-94-007-1338-3
DOI 10.1007/978-94-007-1338-3
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2011931872

© Springer Science+Business Media B.V. 2011
No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

Cover design: VTeX, UAB Lithuania

Printed on acid-free paper

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

Foreword

It is a great pleasure for me to write this foreword about a book that comes out of one of the first research projects funded by ANR, the French National Research Agency. ANR was established by the French government in 2005 to fund research projects, based on competitive schemes giving researchers the best opportunities to realize their projects and paving the way for groundbreaking new knowledge. The role of the Agency is to bring more flexibility to the French research system, foster new dynamics and devise cutting edge-strategies for acquiring new expertise. By identifying priority areas and fostering publicprivate collaborations, the ANR also aims at enhancing the general level of competitiveness of both the French research system and the French economy. The first calls for proposal were launched in early 2005, selection by peer review took place, and ICTER (Information Integrity and Confidentiality for Reconfigurable Technologies), was one of the first selected projects which addressed the issue of security in digital circuits and systems. Since then, many other projects have been submitted to and funded by ANR on closely related topics, this is certainly another measure of the success of ICTER. A few years later, the project results are found to be quite impressive, among which this book is clearly a significant outcome. It will contribute to a better understanding of technologies related to the digital security of electronic devices, which are at the core of information technology as they are key to the trust that we can put in our digital systems. Writing a book of this magnitude is a significant effort; I wish much success to it and congratulate the authors for their achievement. I see it as another proof for ANR of the dynamism of the French research.

Paris, France
2011, March

Professor Bertrand Braunschweig
Head of the Information and Communication Technology
Department of the ANR

Preface

This book is the result of a national project (ICTER) funded by the French National Research Agency (ANR) and involving four research centers (Montpellier, Paris, Lorient, Saint-Etienne) and a private company. When the project started in 2005, very few studies addressed the topic of digital security for reconfigurable architectures including FPGAs (Field Programmable Gate Arrays). But it was already clear that the sustained rate of integrating hardware and software resources in FPGAs would impact future embedded systems, especially in the field of digital security. The complexity of global systems has increased opportunities for designers and users but also resulted in an increase in vulnerabilities. We foresaw this problem and decided to combine efforts to identify the strengths and weaknesses of reconfigurable platforms from the point of view of security. We discovered a world of immense and unique opportunities for research. Our choice immediately focused on a holistic view, taking into account technological, logical, architectural and system levels, and hence, the security pyramid. This concept corresponds to a value chain structure in which the close links between each stage represent a significant gain, but also many potential security flaws.

Since 2005, we have not only been monitoring developments in the field but also been contributing to the state of the art. We have shown that by taking the target technology into account, it is possible to provide innovative techniques to build a system robust enough withstand a large number of attacks. We do not claim to have solved every problem, but we have established some rules and benchmarks that will undoubtedly be used in future applications on FPGAs. We would like to take this opportunity to thank all the contributors to this book, colleagues and PhD and Master students who shared and contributed to the same scientific objectives as we did. A warm thank to Dr. Reouven Elbaz, from Intel company, who friendly spent time to bring comments and remarks on our work. We all participated together in this very exciting project that involved many enthusiastic discussions and culminated in the significant contributions we are happy to share with you here. This project was generously supported by French Research Agency, ANR.[1] In fact, in 2005 ours was

[1] http://www.agence-nationale-recherche.fr/.

among the very first projects to be approved by the ANR. We thank the ANR for having confidence in us and for supporting this work, without this support, this book would not exist.

"Security trends for FPGAs" is designed for all those who would like to upgrade their knowledge in the field of security and digital platforms including reconfigurable FPGAs. We believe you will find many useful technical references and solutions to your problems. We had a lot fun writing this book, we hope you will enjoy reading it just as much.

Montpellier, France — Benoit Badrignans
Paris, France — Jean Luc Danger
Saint-Etienne, France — Viktor Fischer
Lorient, France — Guy Gogniat
Montpellier, France — Lionel Torres

Editors and Chapter Authors

Editors

Benoit Badrignans: Benoit Badrignans obtained his PhD in 2009 at the Laboratory of Informatics, Robotics and Microelectronics, University of Montpellier 2, and his engineering degree at the Polytech'Montpellier. He is currently working for NETHEOS (a private company) in Montpellier, France, as expert engineer in hardware security systems.

Jean Luc Danger: Jean Luc Danger is Professor(in English Professor is the person who has the chair, if this isn't the case, he's probably called a senior lecturer) at TELECOM ParisTech (www.telecom-paristech.fr/). He obtained his engineering degree in Electrical Engineering at the École Supérieure d'Électricité (Paris) in 1981. After 12 years in industrial laboratories (Philips, Nokia), he joined TELECOM ParisTech in 1993. He is currently head of the digital electronic system research group and conducts research in the field of security for embedded systems, configurable architectures and implementation of complex algorithms for telecommunications. Jean Luc Danger is the author of many scientific publications (and patents) on architectures of embedded systems and security, he is co-founder of the company Secure-IC (www.secure-ic.com).

Viktor Fisher: Viktor Fischer is Professor lecturer in Electrical and Computer Engineering at the University Jean Monnet in Saint-Etienne, France, where he has been since 1991, first as a visiting lecturer 6 months a year and from 2006 as a full-time lecturer. He obtained his MSEE and PhD degrees in 1981 and 1991 at the Technical University in Kosice, Slovak republic. From 1981 to 1991, he was a researcher and assistant lecturer in the Department of Electronics, Kosice Technical University. From 1991 to 2006, he worked as a custom chip designer and consultant in the semiconductor industry. His research is in the general area of VLSI design and especially in the use of reconfigurable devices for image processing and data security. He is currently head of the Secured Embedded Systems research group at the Hubert Curien Laboratory (Saint-Etienne, France) and his main research area is true

random number generation and confidential key management for cryptography applications and security of embedded reconfigurable systems. He has published over 60 refereed publications in these areas and is a member of program committees of several leading conferences in the field.

Guy Gogniat: Guy Gogniat obtained his MSEE degree at the University of Paris Sud, Orsay, France, in 1995, and his PhD in electrical and computer engineering at the University of Nice-Sophia, Antipolis, France, in 1997. He is currently a Professor lecturer in electrical and computer engineering at the University of Bretagne-Sud, Lorient, France, where he has been since 1998. In 2005, he spent one year at the University of Massachusetts, Amherst, USA, as an invited researcher, where he worked on embedded system security using Reconfigurable technologies. His work focuses on the design of and methods and tools for embedded systems. He also conducts research in reconfigurable and adaptive computing and embedded system security.

Lionel Torres: Lionel Torres obtained his MSc in 1993 and his PhD 1996 at the University of Montpellier 2. From 1996 to 1997 he worked for ATMEL (a private company) as R&D engineer. From 1997 to 2004 he was associate professor at the University of Montpellier 2, Polytech'Montpellier (Microelectronic design) and LIRMM laboratory. In 2004, he became full Professor and was head of the microelectronics department of LIRMM from 2007 to 2010. He is now deputy head of the Polytech'Montpellier engineering school. His research interests and skills concern reconfigurable computing and system level architecture, with specific focus on security and cryptographic applications. He leads several European, national and industrial projects in this field. He is involved in major conferences and journals and is (co-)author of more that 150 publications.

Contributors

Lyonel Barthe: Lyonel Barthe obtained his engineering degree in electronics at Polytech'Montpellier (University of Montpellier 2), Montpellier, France, in 2009. He is currently a PhD student granted by the DGA at the Laboratory of Informatics, Robotics and Microelectronics, Montpellier, France.

Pascal Benoit: Pascal Benoit obtained his MSc in Microelectronics and Automated Systems at the University of Montpellier, France, in 2001. He obtained his PhD in Computer Engineering at the University of Montpellier in 2004. In November 2004, he joined the Karlsruhe Institute of Technology at the University of Karlsruhe, Germany, where he worked as a scientific assistant. Since September 2005, he as been permanent Associate Professor at the University of Montpellier 2. He teaches the design and programming of embedded systems at the École Polytechnique Universitaire de Montpellier and conducts his research activities at LIRMM, the Montpellier Laboratory of Informatics, Robotics, and Microelectronics which is

a joint University of Montpellier 2 (UM2) and CNRS Department of Information and Engineering Sciences and Technologies research unit. The objective of his research is to design self-adaptive architectures to reduce their power consumption, compensate for their technological variability, and improve their reliability and security. He is involved in major international conferences and projects in the field, and is the (co-)author of more than 60 publications.

Florent Bernard: Florent Bernard is associate professor at the Hubert Curien Laboratory (Saint-Etienne, France) (http://laboratoirehubertcurien.fr). He obtained his BSc in mathematics at the University of Angers and his MSc in cryptography at the University of Grenoble. He obtained his PhD in computer science at the University Paris 8 in 2007, since when he has been working in the SES (Secured Embedded System) team headed by Pr. Viktor Fischer. His research interests are in the area of modeling and evaluating TRNG principles and hardware implementation of cryptographic functions.

Pascal Cotret: Pascal Cotret received his MSc in optics and electronics at Telecom Saint-Etienne, Saint-Etienne, France in 2009, and his BEng Electronic Engineering degree at the University of Glamorgan, Treforest, United Kingdom in 2008. He has been at the Lab-STICC laboratory, University of South Brittany, Lorient, France, since January 2010 where he is preparing his PhD. His research interests include electronic security systems and multiprocessor architectures for reconfigurable technologies such as FPGAs.

Jeremie Crenne: Jermie Crenne received his MSc in electronics and computer science at the University of South Brittany, Lorient, France, in 2008. He is currently preparing his PhD at the Lab-STICC Laboratory, University of South Brittany, Lorient, France. His research interests include electronics security in embedded systems especially with reconfigurable devices.

Amine Dehbaoui: Amine Dehbaoui obtained his engineering degree at the École Nationale Supérieure de l'Électronique et de ses Applications (Paris, France) in 2006, his MSc in Cryptology and Computer Security at the University of Bordeaux I in 2007, and his PhD in Hardware Security at the University of Montpellier II in 2010. He is currently a post-doc at the LIRMM laboratory, Montpellier, France.

Florian Devic: Florian Devic obtained his engineering degree in electronics at Polytech'Montpellier (University of Montpellier 2), Montpellier, France in 2009. He is currently preparing his PhD in the framework of the collaboration between the Laboratory of Informatics, Robotics and Microelectronics of Montpellier and NETHEOS (a private company) in Montpellier, France.

Jean-Philippe Diguet: Jean-Philippe Diguet obtained his MSc and PhD at Rennes University, Rennes, France, in 1993 and 1996, respectively. His thesis addressed the estimation of hardware complexity and algorithmic transformations for high level synthesis. Since 1998, he has been with the University of South Brittany (UBS), Lorient, France, where he started the Design Trotter research project in design space exploration at both algorithmic and systems levels, and since 2004, he has been a

CNRS Researcher with Lab-STICC. He was a Postdoctoral Fellow with IMEC, Leuven, France, where he worked on memory hierarchy decisions for power optimization. From 1998 to 2002, he was Associated Professor at UBS University, Lorient, France. In 2003, he started a technology transfer and cofunded the dixip company in the domain of wireless embedded systems. In 2010 he spent one year at the University of Queensland, Australia where he worked on multiprocessor systems. His current work focuses on a range of topics in the domain of embedded system design: design space exploration extended to heterogeneous embedded realtime systems, environment-aware and selfadaptive HW/SW architectures under QoS, power and performance constraints, RTOS new services for managing reconfiguration, CAD tools for application aware and secured NOC design, and reconfigurable embedded systems as pervasive computing components.

Sylvain Guilley: Sylvain Guilley belongs to the French inter-ministerial body of telecommunication engineers (now called Corps des Mines). He graduated from the École Polytechnique in 1997 and from the École Nationale Supérieure des Télécommunications (ENST) in 2002. In 2002, he obtained his MSc in quantum physics at the École Normale Supérieure (ENS). He obtained his PhD at TELECOM-ParisTech (the new name of ENST) in 2007 on the topic of backend countermeasures against side channel attacks. Since 2002, he has been Associate Professor with the VLSI group (the "SEN" team in the "COMELEC" department) at TELECOM-ParisTech. His research interests are the security of cryptographic hardware (ASIC and/or FPGA) and the specification of provable trusted computing platforms. Sylvain Guilley is co-founder of the TELECOM-ParisTech spin-off Secure-IC, and has been appointed advisor in Secure-IC scientific and strategic consultative boards. He is a member of the IEEE (Institute of Electrical and Electronics Engineers) and of the IACR (International Association for Cryptography Research).

Victor Lomne: Victor Lomne obtained his MSc in Cryptology and Computer Security at the University of Bordeaux I in 2007 and his PhD degree in 2010 from the University of Montpellier 2 (France), at the department of microelectronics of the LIRMM. His research concerns power and electromagnetic Side-Channel Attacks. He is now expert in cryptography of components and hardware security at the components laboratory of the ANSSI (French Network and Information Security Agency).

Philippe Maurine: Philippe Maurine obtained his MSc and PhD in Electronics at the University of Montpellier, France, in 1998 and 2001 respectively. Since 2003, he has been assistant lecturer at the Montpellier Laboratory of Computer Sciences, Robotics and Microelectronics where he works on microelectronics in the engineering program of the University. His current field of research includes timing modeling, statistical timing analysis but also side channel attacks and the design of secure circuits.

Michel Robert: Michel Robert obtained his Master-Engineering degree in 1980 (Polytech'Montpellier) and his PhD in 1987 at the University of Montpellier. From 1980 to 1984 he was in the semiconductor division of Texas Instruments as

R&D engineer. From 1985 to 1990, he was assistant lecturer at the University of Montpellier 2, and LIRMM laboratory. Since 1991 he has been full Professor at the University of Montpellier, Polytech'Montpellier (University of Montpellier 2, France), where he teaches microelectronics in the engineering program. From 2005 to 2010 (LIRMM: www.lirmm.fr), he was director of the Laboratory of Informatics, Robotics and Microelectronics. Michel ROBERT is author or co-author of more than 250 publications in the field of VLSI design. His present research interests are design and modeling of CMOS MP-SOC architectures. He is involved in the IFIP Technical Committee WG 10.5 (Computer Systems Technology, Design and Engineering of Electronic Systems).

Gilles Sassatelli: Gilles Sassatelli is a senior CNRS scientist at LIRMM, which is a CNRS-University of Montpellier II joint research unit. He conducts research in the area of adaptive multiprocessor architectures for embedded systems in the flexible and reconfigurable computing group he leads. He is the author of more than 130 publications in a number of renowned international journals and has given papers at many international conferences. He regularly serves as track or topic chair in major conferences in the field of reconfigurable computing (IEEE FPL, IEEE Reconfig, Worldcomp ERSA, etc.). Most of his research is conducted in collaboration with international partners; over the past 5 years, he has been involved in several national and European research projects including FP6 FET PERPLEXUS STREP.

Romain Vaslin: Romain Vaslin obtained his MSc in electronics and computer science at ESEO, Angers, France, and IRCCyN, Nantes, France, in 2005. He obtained his PhD at the University of South Brittany, Lorient, France in 2008. He is now a research engineer at Thales Communications, Cholet, France. His research interests include electronics security in embedded systems.

Eduardo Wanderley: Eduardo Wanderley received his BEng degree at the Department of Electrical Engineering at UFRN, Natal, Brazil. He received his MSc in electronics and computer engineering at ITA, So Jos dos Campos, Brazil and his PhD at UNICAMP, Campinas, Brazil, both in the Computer Architecture area. He joined LESTER Laboratory, University of South Brittany, Lorient, France in 2006–2007. He is currently with the Federal Institute of Rio Grande do Norte IFRN, Brazil. His research interests include electronics, computer architecture and security in embedded systems.

Contents

1 Introduction and Objectives 1
L. Torres

2 Security FPGA Analysis 7
E. Wanderley, R. Vaslin, J. Crenne, P. Cotret, G. Gogniat, J.-P. Diguet, J.-L. Danger, P. Maurine, V. Fischer, B. Badrignans, L. Barthe, P. Benoit, and L. Torres

3 Side Channel Attacks 47
V. Lomné, A. Dehaboui, P. Maurine, L. Torres, and M. Robert

4 Countermeasures Against Physical Attacks in FPGAs 73
J.-L. Danger, S. Guilley, L. Barthe, and P. Benoit

5 True Random Number Generators in FPGAs 101
V. Fischer and F. Bernard

6 Embedded Systems Security for FPGA 137
B. Badrignans, F. Devic, L. Torres, G. Sassatelli, and P. Benoit

7 Conclusions 189
Benoit Badrignans, Jean-Luc Danger, Guy Gogniat, Viktor Fischer, and Lionel Torres

Glossary 191

Index 195

Chapter 1
Introduction and Objectives

L. Torres

1.1 Motivation

Today there are around 3 billions of cell phone users in the world and 50% of the world population is expected to own a cell phone by the end of 2011. This just shows to what extent electronic mobile devices have already become an integral part of our lives, cell phones being one of the most remarkable examples. Between 2007 and the end of 2011 the equivalent of around 587 billion dollars will be have been exchanged through cell phone transactions. As cell phone users, we really need to have confidence in exchanges with remote servers. What people worry about most when paying for purchases on-line or by cell phone is security. But bank information is not the only sensitive data exchanged with and/or stored on mobile devices. A lot of other personal information can be stolen. Identity theft is a new threat that people are not yet really conscious of. Its easy to steal information from someone and also to steal their identity. In this new connected world, companies are obliged to work on such problems if their customers are to have confidence in their products.

Companies also face new challenges. When a new mobile product comes onto the market, everyone tries to bypass protections and/or restrictions in order to execute specific software procedures. Competitors can spy on technological innovations and hackers can jeopardize their image. The most significant example is the iPhone. Apple applied so many restrictions to its product (phone provider restriction, software execution restriction) that within a few weeks (and even days for the 3G version), the iPhone was jailbroken in order to bypass these restrictions. Today all the protection mechanisms included in software are almost always broken by hackers. In addition, it is difficult to protect a mobile device due to the small hardware and software capabilities of the system. And since the device is out there in the field, owners

L. Torres (✉)
LIRMM—UMR CNRS 5506, University of Montpellier 2, Montpellier, France
e-mail: lionel.torres@lirmm.fr

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_1, © Springer Science+Business Media B.V. 2011

of the system can do almost anything they want with the device, open it, and change the hardware or software.

New reliable solutions are needed to protect both users and manufacturers from 'hackers'. Companies need to protect their intellectual property and business, whereas users are only looking for a secure environment for their data and exchanges. It is now well established that embedded systems are facing increasing attacks since the private digital information embedded in these systems is getting larger every day. The rate of digital data exchanged has also increased exponentially and transferred information has to be kept safe since confidentiality and integrity are mandatory. Embedded systems play an essential role in our society and are already included in many electronic devices from low-end to high-end systems. Security is a serious problem and attacks against these systems are becoming more critical and sophisticated. New solutions are needed to enable the definition of secure embedded systems since current technologies face several challenges and cannot cope with security requirements and performance constraints. Architectures will have to meet high performance requirements, be energy efficient, flexible, tamper resistant and reliable to enable their wide adoption.

FPGAs can address these requirements and provide efficient security primitives. Their characteristics enable the system to prevent attacks or to react when attacks are detected while guaranteeing the necessary energy and computation efficiency. In this book, we present an analysis of current threats against embedded systems and especially FPGAs. We discuss requirements according to the FIPS 140-2 standard in order to build a secure system. This point is of paramount importance as it guarantees the level of security of a system. We also highlight current vulnerabilities of FPGAs at all the levels of the security pyramid (Fig. 1.1). From a design point of view, it is essential to be aware of all the levels to find a comprehensive solution. The strength of a system is defined by its weakest point; there is no advantage in enhancing other means of protection, if the weakest point is left untreated. Many severe attacks considered weak points to escape a complex brute force attack.

Several solutions are described in this book especially at the logical, architectural and system levels to provide a global solution. However, operating system and application levels are beyond the scope of this book as we only deal with hardware solutions here. But it is important to bear in mind that they have to be taken into account for a complete system. A lot of different types of threats have to be considered, any information that leaks from a cryptographic device could be exploited by an attacker. For example, attacks based on power consumption or electromagnetic emissions are possible with low competencies and low cost. Attackers can induce errors during the encryption (or decryption) process in order to collect information concerning secret information, such as cryptographic keys. Therefore cryptographic devices must be protected against fault injection and leakage information. For ASIC, many countermeasures are now well established, but this is not the case for FPGA devices. Only a few academic or industrial studies have been conducted to ensure the confidentiality and integrity of FPGA devices.

The number of possible threats to mobile device is huge. The first step toward a secure environment would be to ensure the data and application cannot be stolen

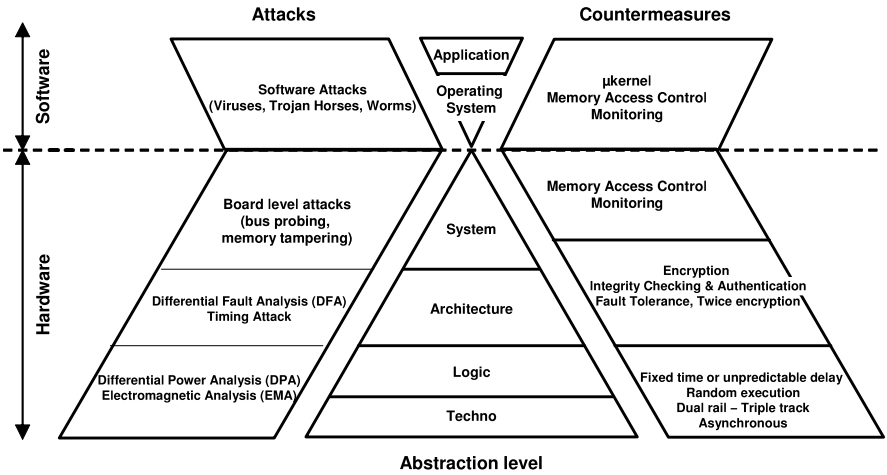


Fig. 1.1 Security pyramid: toward a defense-in-depth

or modified. But is it really possible to have a mobile system with these features? As already mentioned earlier, mobile devices have limited hardware and software resources. In order to tackle the security issues, some new features need to be added to the system. This book details several solutions for secure application execution and application update. New secure schemes are proposed to ensure data confidentiality, integrity and authentication. These new schemes fit the tight requirements of embedded systems (performance, memory footprint, logic area and energy consumption). The cost of different architectures for performance, memory, and energy are estimated. Innovative solutions for remote reconfigurations are also detailed taking into account security when downloading a new bitstream. The replay of an old bitstream in the field is a major threat for embedded systems, this issue is discussed and an original solution proposed.

1.2 Security Model

The security of a device is never perfect: given enough time and money, an attacker can always break into a system. Therefore the security level is sized according to the security objective. To evaluate the robustness of a system, three parameters have to be taken into account:

- The profile of the attacker: this is a combination of different factors: knowledge of the system, its skills (informatics, mathematics and electronics), means (in terms of time and money).
- The different possible methods of attack: the threat varies with the system environment and accessibility. For instance, a satellite is much more difficult to attack physically than are personal security devices like smart cards.

- The value of the protected information: security mechanisms have to be sized according to the value of the information to be protected.

In the particular case of FPGA devices, four different entities are involved: the FPGA vendor who built the FPGA chip, the System Designer (SD) who created an FPGA based system, the IP provider who supplies IP cores used in the system, and finally the system owner who is the end user of the system. In 1991, IBM [1] provided a classification of attackers:

- Level 1: **clever outsider**. These attackers have only limited time and money at their disposal, and only have access to publicly available documentation on the system in question (on Internet, data sheets, software driver source or binary). Their skills are limited to state-of-the-art attacks (mathematics, electronics and informatics) suitable for light weight equipment. They mainly perform known attacks rather than inventing a new form. Two examples of this type of attacker are students or engineers who hack systems during their spare time, for instance, they may tamper with their video game console to crack the game. In the field of FPGA based systems, we consider that they can perform:
  - Trivial attacks with limited equipment, like bus probing, memory dumping,
  - Slightly more elaborate attacks like Simple Power Analysis (SPA) on unprotected cryptographic algorithm implementation,
  - Attacks that require less than the power of a dozen standard current computers.
- Level 2: **knowledgeable insider**. These attackers have access to all the necessary documentation and information (technical documents, C or HDL source codes, electric schematics, bug reports, security vulnerabilities) for the system concerned. They may be former employees of the company that built the system. They generally dispose of considerable means since they may be employed by a competitor. Their level of competence is high regarding the application and the system. They are able to perform the following attacks:
  - Level 1 attacks,
  - Attacks that require confidential information such as encryption keys used in the system,
  - Attacks requiring heavy equipment like Differential Power Analysis (DPA) or fault injection tools,
  - Attacks requiring less than the power of a thousand of modern computers.
- Level 3: **funded organizations**. These attackers are the most powerful imaginable. They have unlimited financial and technical means and time at their disposal. Such attackers are generally big companies, a country, or even criminal organizations (mafia). Typically smart cards used for banking applications are the targets of such attackers. They can apply all known threats such as:
  - Lower level attacks,
  - Attacks requiring less that 2^{80} operations,
  - Invasive attacks aimed at probing or modifying device at transistor and routing level.

All through this book, we consider that FPGA vendors are trustworthy, i.e. they do not intentionally include security vulnerabilities in their chips. The system designer (SD) is also considered to be trustworthy since we tackle the problem of FPGA design security from the point of view of the system designer. IP providers are trusted but do not necessary trust the system designer. Finally system owners are not considered to be trustworthy, for instance companies that provide pay TV do not trust the owner of the system.

We also consider that only level 3 attackers (funded organizations) can perform invasive attacks on the FPGA device, the security mechanisms implemented by the FPGA vendors are subject to the same rule. Therefore below level 3, we consider that the FPGA chip itself is trustworthy.

To summarize, the security level of an application is determined by the power of the attacker, the attack model concerned (for instance local or remote attack) and the value of the information to be protected.

1.3 Organization of the Book

There are five main chapters:

- Chap. 2 provides definitions used for the security model and introduces the vocabulary used throughout the book. In this chapter we also discuss existing attacks at each level of the security pyramid, physical, logical, architectural and system levels, and outline some countermeasures. This chapter provides readers with a first insight into the field of security and FPGAs.
- In Chap. 3 we address the problem of Side Channel Attacks (SCA), mainly differential power and electromagnetic attacks. This chapter is a brief survey of different kinds of general techniques to address SCA.
- These models are then used in Chap. 4 to evaluate robust logic styles in FPGAs. We compare different structures and show that it is possible to propose original logic styles in FPGAs to improve the robustness of FPGAs against SCA.
- In Chap. 5 we discuss how to generate true random number generators in FPGAs, which is essential for most security applications. It is clear that key generation and management are strong points in securing a reconfigurable platform. We propose a set of techniques and several metrics to compare solutions.
- In Chap. 6, we describe how to embed confidentiality and integrity of data processed by reconfigurable platforms at the system level. The threat model is introduced here. Several results are given to illustrate the efficiency of different solutions.

We believe this book provides good overview of different practical techniques that can be used in reconfigurable platforms. However, it is not intended to provide an exhaustive review of all the techniques that can be found in the literature, but rather one way of dealing with security from technology to system for reconfigurable platforms.

Reference

1. Abraham, D.G., Dolan, G.M., Double, G.P., Stevens, J.V.: Transaction security system. IBM Syst. J. **30**(2), 206–229 (1991)

Chapter 2
Security FPGA Analysis

E. Wanderley, R. Vaslin, J. Crenne, P. Cotret, G. Gogniat, J.-P. Diguet, J.-L. Danger, P. Maurine, V. Fischer, B. Badrignans, L. Barthe, P. Benoit, and L. Torres

Abstract Security is becoming since several years a major issue in the domain of embedded systems. Fine grain reconfigurable architectures like FPGAs are providing many interesting features to be selected as an efficient target for embedded systems when security is an important concern. In this chapter we propose an overview of some existing attacks, a classification of attackers and the different levels of security as promoted by the FIPS 140-2 standard. We identify the main vulnerabilities of FPGAs to tackle the security requirements based on the security pyramid concept. We propose a presentation of some existing countermeasures at the different levels of the security pyramid to guarantee a defense-in-depth approach.

2.1 Introduction

Standardized cryptographic algorithms, like AES or RSA, are designed to resist cryptanalysis attacks, such as differential cryptanalysis. Only exhaustive attacks against the cipher key are possible, so the security of these algorithms relies on the length of the cipher key and if it is sufficiently long, such attacks are then impossible. Table 2.1 summarizes current minimum strength recommendations for cryptographic algorithms. In 2011, a minimum of 112 security bits is required for all cryptographic algorithms [5].

Thus cryptography algorithms are mathematically designed to be strong enough for current processing technologies. However, hackers can also attack software or hardware implementations for a lower cost. In such cases, implementation may be the weak point of the encryption process. For example, a new software side channel attack, called Branch Prediction Analysis (BPA) attack, was recently discovered and shown to be practically feasible on popular commodity PC platforms. This shows that a carefully written spy process, run simultaneously with an RSA process, can collect almost all the secret key bits in a *single* RSA signing execution. For this reason, security needs to be considered at different levels, i.e. from the technology to the application. If all these levels and the links between them are not taken into

G. Gogniat (✉)
Lab-STICC—UMR CNRS 3192, Bretagne-Sud University, Lorient, France
e-mail: guy.gogniat@univ-ubs.fr

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_2, © Springer Science+Business Media B.V. 2011

Table 2.1 NIST recommendation for cryptographic algorithms

Date	Min of strength	Symmetric key algorithms	Asymmetric (RSA)	Hash (A)	Hash (B)
2006 to 2010	80	2TDEA*	1024	SHA-1**	SHA-1
		3TDEA*		SHA-224	SHA-224
		AES-128		SHA-256	SHA-256
		AES-192		SHA-384	SHA-384
		AES-256		SHA-512	SHA-512
2011 to 2030	112	3TDEA*	2048	SHA-224	SHA-1
		AES-128		SHA-256	SHA-224
		AES-192		SHA-384	SHA-356
		AES-256		SHA-512	SHA-384
					SHA-512
>2030	128	AES-128	3072	SHA-256	SHA-1
		AES-192		SHA-384	SHA-224
		AES-256		SHA-512	SHA-356
					SHA-384
					SHA-512

Hash (A): Digital signatures and hash-only applications

Hash (B): HMAC, Key Derivation Functions and Random Number Generation. The security strength for key derivation assumes that the shared secret contains sufficient entropy to support the desired security strength. The same remark applies to the security strength for random number generation

*TDEA (Triple Data Encryption Algorithm). The assessment of at least 80 bits of security for 2TDEA is based on the assumption that an attacker has 240 matched plaintext and ciphertext blocks at the most

**SHA-1 has been shown to provide less than 80 bits of security for digital signatures; the security strength against collisions is assessed at 69 bits. The use of SHA-1 is not recommended for the generation of digital signatures in new systems; new systems should use one of the larger hash functions. SHA-1 is included here to reflect its widespread use in existing systems, for which the reduced security strength may not be of great concern when only 80 bits of security are required

consideration, weaknesses can easily and rapidly appear in the devices concerned. Classical implementations of cryptography algorithms and secure embedded systems have been performed on ASICs and processors, but FPGAs are becoming increasingly attractive for cost and performance reasons and should be considered as a new alternative for security issues. As we explain in this chapter, FPGAs provide several key features that are recommended for security. For example when SRAM technologies are used, it is possible to dynamically change the functionality of the system to react to an attack. It is also possible to update some new hardware cryptography cores by remote reconfiguration even when the system has already been used for several years. This helps maintain the security level of a system at a time when cryptanalysis techniques are undergoing constant improvement. FPGAs pro-

vide an appropriate performance level for most embedded systems and, when combined with a processor core, can provide a whole secure solution. In this chapter we present an extensive analysis of FPGAs and their security in order to define the role that this technology could play in future applications.

The rest of the chapter is organized as follows: in Sect. 2.2, to give the reader some general background in the field, we describe the security principles and perform an initial analysis of attacks against FPGAs. In Sect. 2.3, we describe the security requirements for cryptographic modules according to the Federal Information Processing Standard Publication (FIPS PUB 1402) [33]. This point is very important when building a secure system. In Sect. 2.4, we analyze the vulnerabilities of FPGAs according to the security pyramid and describe possible technology, logic, architecture and system levels. In Sect. 2.5, we describe several countermeasures that have been developed and demonstrate how they can be used to build a secure system. Finally in Sect. 2.6, we present a number of conclusions.

2.2 Security Principles and Attacks Against FPGAs

The five main principles on which security is based to ensure the correct execution of a program and the correct management of the communications are:

- *Confidentiality*: only the entities involved in the execution or the communication have access to the data;
- *Integrity*: the message must not be damaged during transfer and the program must not be altered before being executed;
- *Availability*: the message and/or the program must be available;
- *Authenticity*: the entity must be sure that the message comes from the right entity and/or the system must trust the program source code;
- *Non-repudiation*: the entities involved in the execution or the communication must not be able to deny the exchange.

Guaranteeing all these points in a system is intellectually and financially costly. Efforts by attackers to disrupt one of these elements use two different approaches: extracting secret information (or the keys used to codify the information); or disturbing the system. The latter can also be classified in several levels: stop the system; temporarily stop the system; and/or change the functionality of the system (sometimes by opening doors to retrieve secret information).

Attackers are considered as adversaries, with varying abilities and with varying financial means at their disposal, and their attempts to disturb a system must be stopped. Generally, the power of an attacker can be classified using the description provided by IBM [1], as described into Chap. 1.

2.2.1 Hardware Attacks

The main goal of hardware attacks depends on the goal of the attacker. There are generally several objectives. The first is obtaining secret information like cipher keys. The second is causing a breakdown of the system (e.g. denial of service attack). We first describe attacks that aim to obtain secrets, and second, denial of service attacks. Some attacks are difficult to classify, hardware modification of the main memory being one of them since this kind of attack can be considered as a software attack but relies on a hardware technique to modify the memory content. The goal of this attack is to insert a malicious program in the system. A similar attack targets FPGAs by altering the bitstream.

To be able to decrypt information, the attacker needs the cipher key. One way to obtain cipher keys is to listen to side channels. This kind of attack is called a side channel attack and can take several different forms [19]. The best known relies on the power signature of the algorithm [25]. By analyzing the algorithm signature it is possible to infer the round of the algorithm. What is more, differential analysis combined with a statistical study of the power signature can lead to the extraction of the cipher key. However to reach this goal, the attacker has to make certain assumptions about the value of the key. The two methods are called SPA: Simple Power Analysis and DPA: Differential Power Analysis. Similar solutions are also possible using electromagnetic emissions (Differential Electromagnetic Analysis) [3]. Instead of analyzing the power signature, the attacker analyzes electromagnetic signature of the chip. One important aspect is the cost of such attacks. This type of attack is much cheaper than a reverse engineering attack which requires an electronic microscope to study the structure. Temporal analysis or timing attack [24] is another way to obtain cipher keys. The temporal reaction of the system leaks information enables the attacker to extract the cipher key or other secret information such as a password. Like with DPA, the attacker has to make certain assumptions about the information to be extracted, e.g. knowledge of the algorithm, in which case the branch instructions in the program can also help to solve a secret since a timing model of the algorithm can be established. Indeed, timing hypotheses are possible as the program running on the target device is often known. In this case, thanks to statistical studies, information can be extracted.

Fault injection [26] is the last way to obtain secrets through a side channel. However, like reverse engineering, more equipment is required than for the types of attacks described above. The injection of a fault into a system through the memory, for example, corresponds to a modification of a bit (laser or electromagnetic waves). Knowledge of the implementation of the algorithm is crucial to solve a secret. In most cases, the fault is inserted in the last round of an algorithm [26]. This is because the trace of the fault is more visible in the ciphered result. The goal of such hardware attacks is to obtain secret information from the chip. Regular improvements in side-channel attacks have been made in the last ten years and use advanced techniques to extract information. This book provides extensive discussions on the subject.

Denial of service attacks are different and the aim here is to cause a system breakdown. In autonomous embedded systems, power is an essential concern. It is one of the most important constraints on the system. To give an example, with a cell phone or a PDA, an attacker can perform a large number of requests that aim to activate the battery and hence to reduce the life of the system [27, 31]. In wireless communication systems, another type of attack activates the transmitter antenna to obtain the same result (i.e. reducing the life of the system). Increasing the workload of a processor is another way of consuming more battery. Indeed the workload of the system is related to power consumption, so an attacker may try to force the processor to work harder [27, 31]. As a consequence, the lifetime will be affected.

The range of attacks against a system is wide and depends on several parameters: the goal, the budget, and the type system concerned. Hardware attacks are already a serious threat to embedded systems but software attacks are becoming more and more dangerous and need to be recognized and prevented.

2.2.2 Software Attacks

Like in servers and workstations, embedded systems are being increasingly affected by viruses and worms [9]. The difference between a virus and a worm is that a virus requires the help of a human to infect a system and then to spread, whereas a worm does not. A worm is considered to be autonomous. All computer science concepts can be transposed to the embedded system domain. The replacement of a program by a malicious threatens the security of the system. The malicious program may either try to access sensitive data or to shut down the system. Concerning secret data, cipher keys are the most sensitive data as once the attacker knows the cipher keys, he/she has access to all the information in plain. Encrypting memory and protecting cipher keys are classical solutions to these attacks. However protections used in computer science are not appropriate for embedded systems (less computing power and memory). As a result, dedicated solutions for embedded systems are gradually emerging (e.g. bus or program monitoring) [8]. The number of attacks targeting embedded systems is also increasing rapidly. For example, a virus or a worm can be sent to the same system several times to launch the antivirus. Scanning the whole system increases the workload of the processor and thus decreases the battery lifetime which may be critical for autonomous systems. The concept of embedded systems extends the scope of viruses and of worms.

The classification of hardware and software attacks (as depicted in Fig. 2.1) is generic and can be applied to different platforms. Here we focus on attacks against FPGA-based designs which are more hardware oriented, as we explain in the following sections.

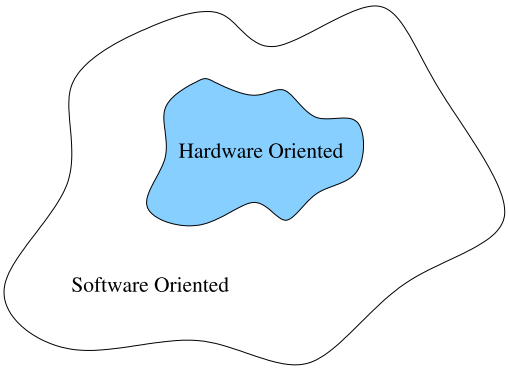


Fig. 2.1 Hardware and software attacks coverage against embedded systems

2.3 Objective of an Attacker

The most common threat against an implementation of a cryptographic algorithm is obtaining a confidential cryptographic key, that is, either a symmetric key or the private key of an asymmetric algorithm. Given that in most commercial applications, the algorithms used are public knowledge, obtaining the key will enable the attacker to decrypt future communications (assuming the attack has not been detected and countermeasures have not been taken) and, which is often more dangerous, to decrypt past communications that were encrypted.

Another threat is the one-to-one copy, or cloning of a cryptographic algorithm together with its key. In some cases this is enough to run the cloned application in decryption mode to decipher past and future communications. In other cases, execution of a particular cryptographic operation with a presumably secret key is—in most applications—the only criterion used to authenticate a party to a communication. An attacker who can perform the same function can attack the system.

Yet another threat is applications in which the cryptographic algorithms are proprietary. Even though such an approach is not common, it is a standard practice in applications such as pay-TV and in government communications. In such scenarios, it is advantageous for an attacker to reverse-engineer the encryption algorithm itself. The associated key might be recovered later by other means (bribery or classical cryptanalysis, for instance). The above discussion generally assumes that an attacker has physical access to the encryption device. Whether that is the case or not depends to a great extent on the application concerned. However, we believe that in many scenarios such access can be taken for granted, either through outsiders or through dishonest insiders.

2.3.1 Security System Using FPGAs

Based on reports by Wollinger et al. [50] and Wollinger and Paar [49], we list the potential advantages of FPGAs in cryptographic applications.

- *Algorithm agility*. This term refers to cryptographic algorithms switching during operation of the targeted application. While algorithm agility is costly with traditional hardware, FPGA can be reprogrammed on the fly.
- *Algorithm upload*. Upgrading fielded devices is conceivable with a new encryption algorithm. FPGA-equipped encryption devices can upload the new configuration code.
- *Architecture efficiency*. In certain cases hardware architecture can be much more efficient if it is designed for a specific set of parameters. One example of a parameter for cryptographic algorithms is the key. FPGA allows this type of architecture, and enables optimization using a specific set of parameters. Depending on the type of FPGA, the application can be completely or partially modified.
- *Resource efficiency*. The majority of security protocols are hybrid protocols that require several algorithms. As they are not used simultaneously, the same FPGA device can be used for both through runtime reconfiguration.
- *Algorithm modification*. There are applications that require modification of standardized cryptographic algorithms.
- *Throughput*. General purpose microprocessors are not optimized for rapid execution. Although typically slower than ASIC implementations, FPGA implementations have the potential to run much faster than software implementations (as with a processor).
- *Cost efficiency*. There are two cost factors that have to be taken into consideration when analyzing the cost efficiency of FPGAs: the cost of development and the unit price. The cost of developing an FPGA implementation for a given algorithm is much lower than for an ASIC implementation. Unit prices are not high compared with the cost of development. However, for high-volume applications (more than one million circuits) ASIC is usually the most cost-efficient choice.

FPGAs obviously have some interesting features and should not be discarded for security applications. To analyze the problem of FPGA security, it is important to define the model of computation to be used and the areas to be protected. There are three possibilities: The first is considering the FPGA and its surrounding (normally a processor and memory) as a trusted area; the second is restricting the trusted area to the FPGA itself; and the third is when certain functional parts inside the FPGA are considered to be trustworthy and others are not.

2.3.1.1 FPGA Based Security Models

In the context in which the FPGA and its environment is considered as a trusted area (Fig. 2.2), the main element involved in the system security is the I/O interface. In this case, the data entering or leaving the system need to be protected (for example, confidentiality and authentication). In fact three interfaces can be used depending on the boundary of the security perimeter. The first is related to the I/Os of the system. In a secure execution context no information should leak from this interface so all the data has to be encrypted. The second is related to the FPGA configuration file if remote configuration is possible. In this case, it is essential to protect the

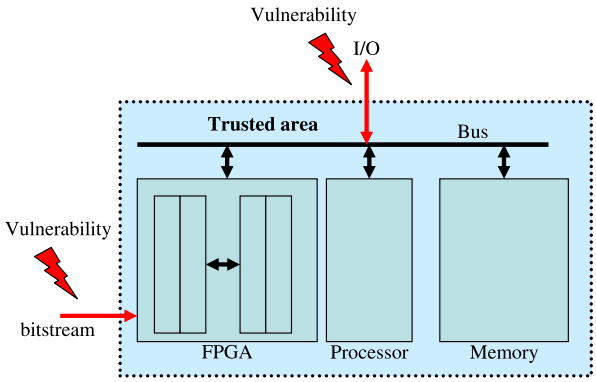


Fig. 2.2 FPGA, processor and memory trusted area: model one for FPGA-based secure systems

bitstream by encrypting it. If all the configurations are within the trusted area, they can be in clear form. The third interface is related to all the critical information dealing with the security of the system, for example the transfer of a new key or a new certificate. In this case, the interface needs to be different from the I/O one in order to increase the security of the system; this interface should also be protected. Generally authentication mechanisms are needed to ensure that only an authorized party can send new data through this interface.

In the second context, the FPGA is considered as a trusted area but its environment is not (Fig. 2.3). For such a model, in addition to protecting the I/O interfaces with the system as a whole, it is necessary to protect the communication inside the system in a per-block (Memory, Processor, FPGA) granularity. In this case, several techniques need to be considered in order to provide authentication, confidentiality and integrity verification. Furthermore, as the number of communications over the bus is generally critical, very efficient and optimized cryptography resources

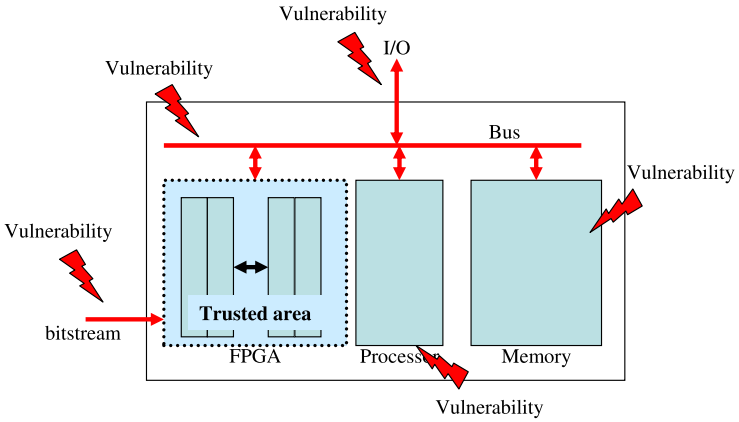


Fig. 2.3 FPGA trusted area: model two for FPGA-based secure systems

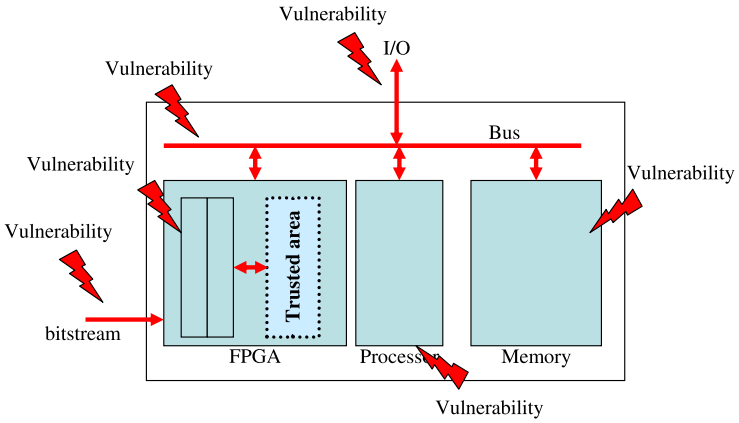


Fig. 2.4 Modules of FPGA trusted area: model three for FPGA-based secure systems

are needed. The latency of the exchanges is of paramount importance and needs to be tackled. In the second context, protecting the FPGA configuration is also an important issue.

Finally, in the third context, the FPGA itself contains regions that are trusted and regions that are not (Fig. 2.4). In such a situation, only the configurations (i.e. bitstreams) that compose the FPGA functionality need to be encrypted. FPGA's trusted area needs to be protected, but at this granularity, fine aspects of bitstreams have to be considered. This solution involves the same challenges as the previous ones since data exchanged within the FPGA but also exchanged with external resources need to be protected, but the solutions should lead to a very low overhead so as to not penalize the execution of the whole system. This model is clearly the most challenging one to design.

2.3.1.2 Threats Against FPGAs

Before building a solution it is essential for designers to clearly define the execution context and the kinds of threats they will be facing.

In this section, we describe several types of attacks against FPGAs. However, some of them will be discussed in detail in subsequent sections when we deal with FPGAs' vulnerabilities and countermeasures.

Black Box Attack—In this type of attack, the intruder sends all possible input combinations and with the results he/she obtains, he/she may be able to reverse-engineer a chip. In practice, this type of attack is difficult to perform on complex systems.

Read-Back Attack—Read back attacks are based on the ability to read the FPGA configuration, usually using the JTAG plug. This feature is provided in most FPGAs to promote debugging capabilities. The aim of the attack is to read the configuration

of the FPGA through the JTAG or programming interface to obtain secret information. Recently FPGAs vendors have considerably improved their devices to increase the level of protection.

Cloning of SRAM FPGAs—SRAM FPGA based systems normally store the configuration file in a non-volatile memory outside the FPGA. In such a situation, an eavesdropper can easily retrieve the configuration file flowing through the port, and possibly clone the same design in other FPGAs. The only possible way to protect the system in this case is to encrypt the bitstream. FPGAs vendors provide this possibility in their most recent devices.

Physical Attack against SRAM FPGAs—The goal of an attack targeting the physical layer of an FPGA is to investigate the chip design in order to obtain secret information by probing points inside the FPGA. These attacks target the parts of the FPGA that are not available through the normal I/O pins. Using instruments based on focused ions (FIB), for example, the attackers can inspect the FPGA structure and retrieve the design or the keys. Such attacks are hard to implement due to the complexity of the equipment required. Moreover, some technologies, like Antifuse FPGAs and Flash FPGAs, which have their own limitations, can make attacking even harder.

Side-Channel Attacks—In this case, the physical implementation of the systems is used to leak information like energy consumption, execution time and electromagnetic fields. By observing these phenomena, an attacker can obtain the power, time and/or electromagnetic signatures of the system, which, in turn, can reveal secrets concerning the underlying implementation. Gathering such signatures is one step in the problem. In fact, the data obtained still has to be processed to obtain the desired results. Very sophisticated techniques have been developed in the last few years that require few measurements to attack a system.

Data analysis—In fact, the data acquired by read back attacks, as well as those from side channel attacks, are considered as noise. The fact that an attacker possesses this information does not imply that he/she will be able to obtain the original design running in the FPGA, but nevertheless makes this possible.

Reverse-engineering is the work done after the bitstream has been obtained, for example, when it is necessary to discover the data structure used by the manufacturer to codify the configuration of the FPGA. Reverse engineering is not limited to bitstreams but can also be achieved by observing bus activities during program execution in a softcore processor implemented in the FPGA. Many reverse engineering attempts on FPGAs succeed, and normally the manufacturers use a disclosure term to morally refrain the attackers, which is not sure at all.

Leakage data is processed using techniques like *Simple Power Analysis* (SPA) or *Differential Power Analysis* (DPA). Normally the aim of these approaches is to identify energy consumption patterns similar to the ones obtained in the known execution pattern of a cryptographic algorithm. Then, finding the key is only a question of time and of the quantity of statistics obtained.

2.4 Security Requirements for Modules

The security of systems used to protect sensitive information is provided by cryptographic modules. Security requirements for cryptographic modules are specified in the Federal Information Processing Standard Publication (FIPS PUB 140-2) [33]. These requirements are related to the secure design and implementation of a cryptographic module.

2.4.1 Security Objectives

Security requirements for cryptographic modules are derived from the following high-level functional security objectives:

- To employ and correctly implement the approved security functions for the protection of sensitive information.
- To protect a cryptographic module from unauthorized operation or use.
- To prevent the unauthorized disclosure of the contents of the cryptographic module, including plaintext cryptographic keys and other critical security parameters (CSPs).
- To prevent the unauthorized and undetected modification of the cryptographic module and cryptographic algorithms, including the unauthorized modification, substitution, insertion, and deletion of cryptographic keys and CSPs.
- To provide indications of the operational state of the cryptographic module.
- To ensure that the cryptographic module performs properly when operating in an approved mode of operation.
- To detect errors in the operation of the cryptographic module and to prevent the compromise of sensitive data and CSPs resulting from these errors.

While the security requirements specified in the FIPS 140-2 standard are intended to maintain the security provided by a cryptographic module, conforming with this standard is necessary but is not sufficient to ensure that a particular module is secure. The operator of a cryptographic module is responsible for ensuring that the security provided by the module is sufficient and acceptable to the owner of the information that is being protected, and that any residual risk is acknowledged and accepted.

Similarly, the use of a validated cryptographic module in a computer or telecommunication system is not sufficient to ensure the security of the whole system. The overall security level of a cryptographic module must provide the level of security that is appropriate for the security requirements of the application and of the environment in which the module has to be used, and for the security services that the module has to provide.

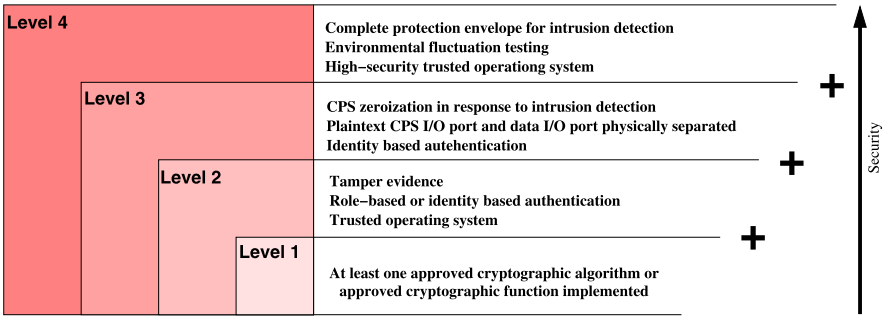


Fig. 2.5 Security requirements in four security levels, according to the FIPS 140-2 Standard

2.4.2 Security Levels

The standard defines four qualitative levels of security—Level 1 to Level 4 (Fig. 2.5). These four increasing levels of security enable cost effective solutions that are appropriate for different degrees of data sensitivity and different application environments.

Security Level 1 requires only the use of an approved algorithm or security function. It allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an unevaluated operating system. No specific physical security mechanisms are required in a Security Level 1 cryptographic module beyond the basic requirement for production-grade components. *Security Level 2* enhances the physical security of the module by adding the requirement for tamper evidence and a role based authentication of an operator. A trusted (verified) operating system with an approved degree of security should be used starting from Level 2. In *Security Level 3*, the security mechanisms should detect and respond to attempts at physical access, unauthorized use or modification of the cryptographic module. The tamper detection circuitry should "zeroize" all plaintext CSPs when a physical intrusion is detected. At this security level, the role based authentication from Level 2 is replaced by identity based authentication. At Level 3, the I/Os of plaintext CSPs should be performed using ports that are physically separated from other data ports. *Security Level 4* is the highest security level. The module should be enclosed in a complete protection envelope. Intrusion into the module should have a very high probability of being detected and the module should detect fluctuations in environmental conditions, namely voltage and temperatures that are outside the normal operating range.

2.4.3 Security Requirements

The security requirements specified in the FIPS 140-2 standard cover 11 areas related to the design and implementation of a cryptographic module. These

areas include cryptographic module specifications; module ports and interfaces; roles, services, and authentication; finite state models; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self tests; and design assurance. The last area concerned with the mitigation of other attacks has not yet been tested but the vendor is required to document implemented controls (e.g. differential power analysis). While most of these areas are related to the technology used (FPGAs in our case), some only concern design methodology (e.g. finite state models). Below, we analyze only those design and implementation areas that are directly related to the application of FPGAs in security modules. These areas include:

- Cryptographic module specification
- Cryptographic module ports and interfaces
- Physical security
- Operational environment
- Cryptographic key management
- EMI/EMC
- Self-tests

Cryptographic module specification deals with a combination of hardware, software and firmware means used to implement cryptographic functions and protocols within a defined cryptographic boundary. A cryptographic module should implement at least one approved security function used in an approved mode of operation. A cryptographic boundary should consist of an explicitly defined perimeter that establishes the physical bounds of a cryptographic module. If a cryptographic module consists of software or firmware components, the cryptographic boundary should contain the processor(s) and other hardware components that store and protect the software and firmware components. Hardware, software, and firmware components of a cryptographic module can be excluded from the security requirements if it is demonstrated that these components do not affect the security of the module.

Cryptographic module ports and interfaces—Two hierarchical levels of I/O ports are defined by the standard: physical ports and logical interfaces. The module should have at least four logical interfaces (data input, data output, control input and status output) that can share the same physical port. Data input and output are used to transfer plaintext and ciphertext data and plaintext (only up to Security Level 2) and ciphertext CSP. Starting from Security Level 3, the I/O port for plaintext CSP should be physically separate from other data ports. All data output via the data output interface should be prevented when an error state exists and during self tests. The control input interface serves to provide commands and signals, and to control data (including function calls and manual controls such as switches, buttons, and keyboards). The status output interface enables the design to send signals, indicators, and status data (including return codes and physical indicators such as Light Emitting Diodes and displays).

Physical security—A cryptographic module is protected by physical security mechanisms in order to restrict unauthorized physical access to the contents of the module and to avoid unauthorized use or modification of the module (including

substitution of the entire module) when installed. All hardware, software, firmware, and data components within the cryptographic boundary should be protected. Physical security requirements are specified for three defined physical embodiments of a cryptographic module:

- *Single-chip cryptographic modules* use a single integrated circuit (IC) chip as a stand alone device. Examples of single chip cryptographic modules include smart cards with a single IC chip. Although single chip cryptographic modules are the most vulnerable to side channel attacks, they are usually used in a hostile environment. Single chip modules based on FPGAs have not been used up to now, since they involve the use of a non-volatile technology.
- *Multi-chip embedded cryptographic modules* are physical devices in which two or more IC chips are interconnected. If possible, they should be embedded in an opaque enclosure. Examples of multi-chip embedded cryptographic modules include adapters and expansion boards.
- *Multi-chip stand-alone cryptographic modules* are physical embodiments in which two or more IC chips are interconnected and the entire enclosure is physically protected. Examples of multi-chip, stand alone cryptographic modules include encrypting routers or secure radios.

Operational environment—The operational environment of a cryptographic module refers to the management of the software, firmware, and/or hardware components required for the module to operate. The operational environment can be non-modifiable (e.g. firmware contained in ROM, or software contained in a computer with I/O devices disabled), or modifiable (e.g. firmware contained in RAM or software executed by a general purpose computer). An operating system is an important component of the operating environment of a cryptographic module.

Cryptographic key management—The security requirements for cryptographic key management encompass the entire life cycle of cryptographic keys, cryptographic key components, and CSPs used by the cryptographic module. Key management includes random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization. Secret keys, private keys, and within the cryptographic module, CSPs should be protected from unauthorized disclosure, modification, and substitution. Inside the cryptographic module, public keys should be protected against unauthorized modification and substitution. A cryptographic module may use random number generators (RNGs) to generate cryptographic keys and other CSPs internally. There are two basic classes of generators: deterministic and nondeterministic. A deterministic RNG consists of an algorithm that produces a sequence of bits from an initial value called a seed. A nondeterministic RNG produces output that depends on some unpredictable physical source that is beyond human control. There are no FIPS Approved nondeterministic random number generators. If intermediate key generation values are sent outside the cryptographic module, the values should be sent either (1) in encrypted form or (2) under split knowledge procedures. Key establishment can be performed by automated methods (e.g. use of a public key algorithm), manual methods (use of a manually-transported key loading device), or a combination of automated and

manual methods. Cryptographic keys stored inside a cryptographic module should be stored either in plaintext or encrypted. Plaintext secret and private keys should not be accessible to unauthorized operators from outside the cryptographic module. A cryptographic module should associate a cryptographic key (secret, private, or public) stored inside the module with the correct entity (e.g. the person, group, or process) to which the key is assigned. A cryptographic module should provide methods to zeroize all plaintext secret and private cryptographic keys and CSPs inside the module. Zeroization of encrypted cryptographic keys and CSPs or keys otherwise physically or logically protected inside an additional embedded validated module (meeting the requirements of this standard) is not required.

Electromagnetic Interference Compatibility (EMI/EMC)—Cryptographic modules should meet EMI/EMC requirements. While the metallic enclosure of the multichip cryptographic module enables these requirements to be met quite easily, standard single chip modules are difficult to design.

Self-Tests—A cryptographic module should perform power up self tests and conditional self tests to ensure that the module is functioning properly. Power up self tests should be performed when the cryptographic module is powered up. Conditional self tests should be performed when an applicable security function or operation is invoked (i.e. security functions for which self tests are required). If a cryptographic module fails a self test, the module should enter an error state and output an error indicator via the status output interface. The cryptographic module should not perform any cryptographic operations while in an error state. All data output should be inhibited when an error state exists. A cryptographic module should perform the following power up tests: cryptographic algorithm test, software/firmware integrity test, and critical functions test. It can also perform the following conditional self tests: pair-wise consistency test, software/firmware load test, manual key entry test, continuous random number generator test, and bypass test.

2.4.4 Security Policy

A cryptographic module security policy should consist of a specification of the security rules, under which a cryptographic module should operate, including the security rules derived from the requirements of the standard and the additional security rules imposed by the vendor. There are three major reasons that a security policy is required:

- It is required for FIPS 140-2 validation.
- It allows individuals and organizations to determine whether the cryptographic module, as implemented, satisfies the stated security policy.
- It describes the capabilities, protection, and access rights provided by the cryptographic module, allowing individuals and organizations to determine whether it meets their security requirements.

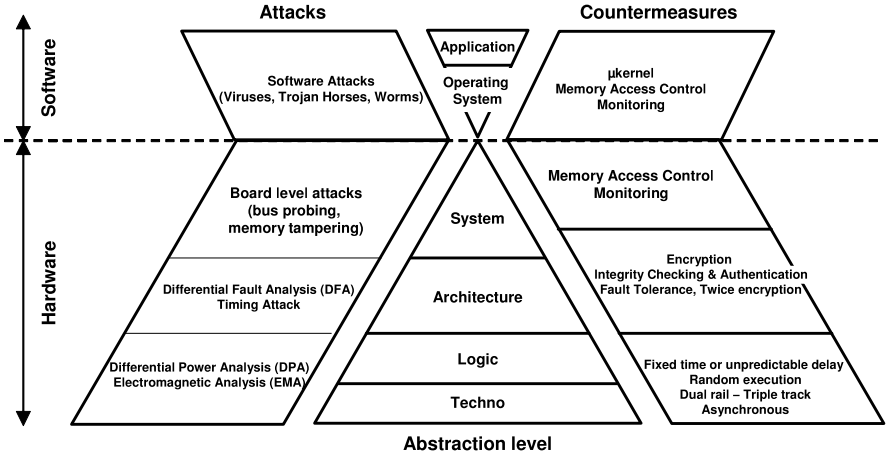


Fig. 2.6 Security pyramid: toward a defense-in-depth

2.5 Vulnerabilities of FPGAs

When dealing with security, it is important for the designer to not disregard any parts of the security barriers. The strength of a system is defined by its weakest point; there is no reason to enhance other means of protection if the weakest point remains untreated. To address this fundamental issue, the different hierarchical levels of a design (from application to technological levels) must be reviewed. Each level has specific hardware or software weaknesses, so specific mechanisms need to be defined in order to build a global secure system (i.e. defense in depth). Depending on the requirements and the security to be reached, several levels have to be considered. Thus it is important to clearly define the security boundaries for the system to be protected. In this chapter, we tackle this point by defining the security pyramid that covers the levels of security. In the following sections we address the different levels from technological to system levels as highlighted in Fig. 2.6 and slightly discussed into the Chap. 1. Operating system and application levels are beyond the scope of this chapter as we only deal with hardware solutions. However it is important to bear in mind that they need to be taken into consideration for a complete system. A lot of threats have to be considered, all information that leaks from a cryptographic device can be exploited by an attacker. For example, attacks based on power consumption or electromagnetic emission can be performed with low competencies and at low cost. Furthermore, attackers can cause errors during the encryption (or decryption) process aimed at to obtaining secret information, such as cryptographic keys. Therefore cryptographic devices must be protected against fault injection and leakage information. Many countermeasures are now well established for ASIC, but this is not yet the case for FPGA devices. Only a few academic or industrial studies have been conducted to ensure the confidentiality and integrity of FPGA devices. Nevertheless, this topic has attracted a lot of attention in the last few years and major progress is underway.

2.5.1 Technological Level

The technological level corresponds to the device and mainly concerns tamper evidence or resistance. Many authors have targeted the technological level but mainly for ASIC-based designs. However in [50] and [49], the authors performed an in-depth analysis of attacks against FPGAs at the technological level. Several technologies are possible for FPGAs, the most widespread being SRAM, Flash and Antifuse. Each technology has advantages but also some limits. Secured ASICs frequently incorporate mechanisms able to detect an attempt by an attacker to make an invasive attack, for example by removing package sealing and by inserting probes at appropriate points in order to obtain cipher keys. At chip level, the mechanism could be distributed sensors to check the package has not been removed. These secured packages are also suitable for FPGA devices. However at die level, FPGA users are limited by the capabilities of the device, they cannot add special security mechanisms such as analog sensors. The level of protection provided by FPGAs technologies is an interesting metric to identify the studies required to improve the security level. In the IBM Systems Journal, Abraham et al. [1] defined the security levels for modern electronic systems.

- *Level 0 (ZERO)*—No special security features added to the system. It is easy to compromise the system with low cost tools.
- *Level 1 (LOW)*—Some security features in place. They are relatively easily defeated with common laboratory or shop tools.
- *Level 2 (MODLOW)*—The system has some security against non-invasive attacks; it is protected against some invasive attacks. More expensive tools are required than for level 1, as well as specialized knowledge.
- *Level 3 (MOD)*—The system has some security against non-invasive and invasive attacks. Special tools and equipments are required, as well as some special skills and knowledge. The attack may be time consuming but will eventually succeed.
- *Level 4 (MODH)*—The system has strong security against attacks. Equipment is available but is expensive to buy and operate. Special skills and knowledge are required to use the equipment for an attack. More than one operation may be required so that several adversaries with complementary skills would have to work on the attack sequence. The attack could fail.
- *Level 5 (HIGH)*—The security features are very strong. All known attacks have failed. Some research by a team of specialists is necessary. Highly specialized equipment is necessary, some of which might have to be specially designed and built. The success of the attack is uncertain.

According to this classification, it is possible to specify a general security level for existing FPGA technologies and ASIC circuits. These levels are not fixed and depend on the factory and the type of circuit (several families may be processed in the same factory and some of them may be highly security efficient, like military families). Table 2.2 lists the security levels of existing technologies, especially FPGA circuits.

Table 2.2 Security level of classical integrated circuits

Integrated circuit	Security level
SRAM FPGA	0
ASIC gate array	3
Cell-based ASIC	3
SRAM FPGA with bitstream encryption	3
Flash FPGA	4
Antifuse FPGA	4

SRAM FPGAs need a bitstream transfer from the root EEPROM at power up (due to the configuration memory that is an SRAM volatile memory). Consequently, it is easy for a hacker to read the bitstream during the transfer using a simple probe.

Thus unprotected SRAM FPGAs are not efficient for safe design. However, with a bitstream encryption it is possible to considerably improve the security level since the security weakness is overcome. SRAM FPGAs have a good resistance against some attacks like power analysis. Although ASICs are often considered to be a secure technology, they are actually relatively easy to reverse engineer. Because, unlike FPGAs, ASICs do not have switches, and it is thus possible to strip the chip and copy the complete layout to understand how it works. Methods to reverse engineer ASIC exist. The cost of reverse engineering is high since the tools required are expensive and the process is time consuming. It is not a simple process and the security level is 3 for such devices. Contrary to ASICs, FPGAs, like antifuse or flash, are security efficient since they are based on switches. With these FPGAs, no bitstream can be intercepted in the field (no bitstream transfer, no external configuration device). In the case of antifuse FPGAs, the attacker needs a scanning electron microscope to identify the state of each antifuse. Nevertheless, the difference between a programming and a non-programming antifuse is very difficult to see. Moreover, such analysis is intractable in a device like Actel AX2000 that contains 53 million antifuses and, according to Actel (http://www.actel.com/products/solutions/security/), only 25% (on average) of these antifuses are programmed. For flash FPGAs, there is no optical difference after configuration, so invasive attacks are very complex. The same advantages are cited by QuickLogic to promote their flash FPGAs with ViaLink technology. Even if the antifuse and the flash FPGAs are very security efficient, they can only be configured (or programmed) once, so they are not reconfigurable devices. Furthermore as defined by the FIPS 140-2 standard, for security levels 3 and 4, it has to be possible to zeroize all the secret information, which is not possible with technologies like flash and antifuse. The system built with these devices is thus not flexible. If the designer wants a reconfigurable device, he should choose an SRAM FPGA. Moreover, the capacities of the SRAM FPGAs are the highest for FPGA devices. The market share of SRAM FPGAs is more than 60% (just counting the two leading companies Xilinx (http://www.xilinx.com) and Altera (http://www.altera.com)). Further research is required to improve the security level of such FPGAs and particularly to improve bitstream encryption. In recent years, FPGAs vendors have been tackling this point

in order to provide their customers with efficient solutions to encrypt the SRAM FPGA bitstream. However, they still have some drawbacks and could be further improved by taking the latest innovations of these FPGAs into account.

2.5.2 Logical Level

It is now widely recognized that the Achilles' heel of secure applications, such as 3DES and AES ciphering algorithms, is their physical implementation. Among all the potential techniques to retrieve the secret key, side channel attacks are worth mentioning. Although there are a lot of different types of side channel attacks, the DPA attack is considered to be one of the most efficient and most dangerous since it requires few skills and little equipment to be succeed. Thus at the logical level, all attacks that are possible on ASICs, like side channel attacks and fault attacks, are reproducible in FPGAs. Secret leakage can even be amplified, which makes the attack easier. At first sight, FPGAs appear to be less robust than ASICs. This could have different causes:

- The FPGA structure has heavy loaded wires made up of long lines or lines segmented by pass transistors. As power consumption is proportional to the capacitance load, this makes measuring power consumption more easy.
- The designers often take advantage of pipelining in FPGAs, as the DFF (D Flip Flop) is "free" in every cell. The DFF is not only used as a power contributor to power consumption in CMOS ASICs but as predictor (for correlation attacks) of relevant key dependent consumption for the attack strategies. The FPGAs have very rapid (for performances and fighting metastability) and high power consuming DFFs at their disposal but they also drive logic that could be predictable.
- The use of pass transistors generates a power consumption expression that varies partly in Vdd^3 and not only in Vdd^2, and as the CMOS gates are just under the conduction threshold, this helps make measuring the power consumption easier in FPGAs.

All these reasons make FPGAs more vulnerable to attacks since they are based on variations in power (analyzed for passive attacks and provoked for fault attacks) on cells where the key is involved. When analyzing power consumption during a ciphering operation, peaks are clearly discernible in the acquisition trace at every clock period. An efficient attack on FPGA is based on predicting the transitions of relevant DFF or logic driven by these DFFs, where there could be contribution to or leak from the key. For instance, it is possible to predict logic states at the S-box outputs during the first and last round of symmetric key algorithms. Many attacks on FPGA implementations have been reported in the last five years. For example, in [34], an SPA was successfully used in an unprotected implementation of an elliptic curve cryptographic algorithm. Furthermore as underlined in [34], attacks on FPGAs allow a hacker to:

- Evaluate the intrinsic resistance or vulnerability of this device class;

- Assess the resistance or vulnerability of the algorithms, executed on a real concurrent platform. Compared to a simulation, measurements made on an emulating device are indeed expected to be closer to the final projection in an ASIC.

In [42] and [39], the authors describe successful correlation power attacks (CPAs) against DES and AES implementations programmed into FPGA. The attacks scenario does not differ from that of ASICs (such as smart cards). Unsurprisingly, in all cases, the attacks were just as successful as when carried out against hardwired devices. As a consequence, there is a real incentive to devise countermeasures that take the architecture of FPGAs into account. It is important to come up with innovative solutions based on accurate knowledge of the FPGAs' internals in order to counter these attacks. Current FPGAs vendors have not yet taken up this challenge.

2.5.3 Architecture Level

Logical errors can be used to perform attacks at algorithmic level. A simple example is an attack against RSA algorithm implementation. RSA is based on modular exponentiation, decryption of a ciphered message C follows the following formula:

$$M = C^E \bmod N,$$

where C is the ciphered message, E the private exponent, N the public modulus, and M the unciphered message

The usual way to perform this calculus is the square and multiply algorithm, for either software or hardware implementation.

In the original algorithm, multiplication is carried out only if the exponent bit concerned is 1. So a dummy operation can be inserted in order to have a constant computation time. Supposing that an attacker can create an error at a precise algorithm step, he/she could easily guess the decryption key, which is secret information. To succeed, the attacker must create an error during the first multiplication and only the first; at the end of the algorithm, if the result is wrong, the multiplication was not a dummy one, so the first bit of the exponent is 1. If the result is correct, the multiplication was a dummy one, so the first bit of the exponent was 0. The other exponent bits can be discovered by repeating this scheme for all the multiplication steps. There are many ways to create an error during a computation. A simple way is to increase clock frequency above the maximum allowed by the attacked device. Uncommon temperature, voltage supply or clock glitch can also cause computation errors. With laser or focus ion beams, errors can be created in specific areas in the circuit. This type of fault injection requires specific expensive equipment but can provide better results. In the previous example, attackers do better by introducing errors only in the multiplier area, consequently other operations will not be affected. Other types of fault analysis exist, [16] deals with such attacks on smart card implementation of AES symmetric cipher. Of course this attack can target ASIC as well as FPGA platforms. However, in the future it will be interesting to see if FPGA structure could provide specific countermeasures. This point was judiciously exploited

by [39] to explore the advantages and limits of pipelining techniques from a security point of view. This architectural aspect (along with scheduling and resources allocation) is indeed a new dimension when refining a software code into an RTL description. FPGAs are appropriate tools for the rapid comparison of different architectures. At the architectural level, one basic block to consider is the processor. In the following section an example is given based on 32-bit processor and results obtained with SCA.

2.5.3.1 Processor Level

The MicroBlaze and the Nios II are 32-bit soft-core processors designed and supported by Xilinx and Altera respectively, for their FPGAs. Because of the growing number of embedded systems involving applications with security services, it is necessary to analyze potential weaknesses of these architectures used in most of the modern FPGAs.

The MicroBlaze and Nios II implement both a classic RISC Harvard architecture exploiting the Instruction Level Parallelism (ILP) with a 5-stage pipeline: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MA), Write Back (WB). All pipe stages are hooked together with a set of registers, commonly named pipeline registers. They play the main role of carrying data and control signals for a given instruction from one stage to another.

As a case study, an evaluation was conducted on the MicroBlaze with the Data Encryption Standard. The processor was programmed in ANSI C code on a Xilinx Spartan-3 Starter Kit board (XC3S1000 FPGA). Without loss of generality, the attacks were performed at the end of the first encryption round, when the left part of the plaintext message and the result of the Feistel function are XORed. The result of this operation contains a partial information about the cipher key, and thus, is a potential point of attack.

Critical instructions are detailed in the following list:

- **LWI R4, R19, 44**; the result of the Feistel function is loaded from data memory into register R4
- **LWI R3, R19, 40**; the left part of the message is loaded from data memory into register R3
- **XOR R3, R4, R3**; the result of the XOR operation between R3 and R4 is stored into register R3
- **SWI R3, R19, 32**; the content of R3 is stored into data memory

According to these instructions and to the chosen model of attacks, the sensitive data are handled during the XOR and the SWI instructions. During the execution of these instructions (in Fig. 2.7), we observe that the sensitive data are unprotected during the EX, MA, and WB stages.

A Differential ElectroMagnetic Analysis (DEMA) was conducted on the proposed DES software implementation [6]. The secret key was discovered with less than 500 electromagnetic traces. Besides, the most interesting result is the effect

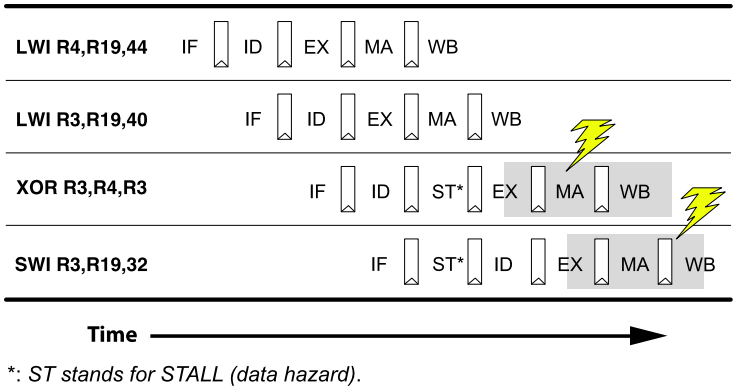


Fig. 2.7 The 5-stage pipeline during the execution of critical instructions

of the pipelining technique. Figure 2.8 illustrates the DEMA obtained for the first sub-key (50,000 electromagnetic traces were collected to emphasize the impact of this hardware feature). In the picture, the black curve refers to the correct sub-key, while the others correspond to the wrong sub-key hypotheses (the guessed sub-key is characterized by the highest amplitude).

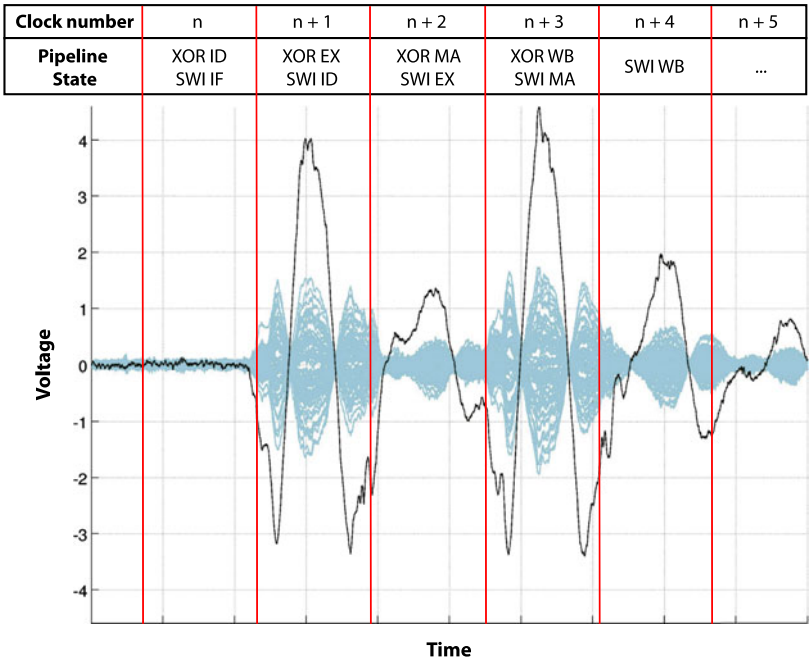


Fig. 2.8 DEMA traces obtained for the first sub-key of the DES software implementation with the MicroBlaze

The margin is basically defined as the minimal relative difference between the amplitude of the differential trace obtained for the guessed sub-key (black curve) and the amplitude of the differential traces obtained for the other sub-keys (other curves). This one reaches more than 50% for the correct sub-key during several time periods, which underlines the considerable vulnerability of a pipelined architecture.

The conclusion to be drawn from the above investigation seems fairly straightforward: the pipelined datapath of embedded microprocessors for FPGAs is a critical security issue.

2.5.4 System Level

At this level, we consider the system as a set of communicating blocks. Some are secure, and the main focus is communication between the modules. One of the threats at the system level is the "man in the middle" attack, in which an adversary eavesdrops on the communication channel. The security of the blocks depends on the cryptographic algorithm used to cipher the information. The problem is ensuring that the communication is secure in an unsecured channel. In addition, guaranteeing security has a cost and the cost of communication may be increased by the use of security dedicated modules integrated in the original design. The solutions described below are some of the recent techniques described to address this problem. Some of them are orthogonal in relation to others, meaning that we simultaneously can use a set of techniques. As mentioned in previous sections, in an FPGA context, the configuration bitstream is one potential weakness in secure applications. This bitstream can be used to perform various types of attacks and needs to be considered at the system level in order to define a global solution.

- *Bitstream retrieval*—The first step is to retrieve the entire bitstream from the system. The easiest way is to use the read-back capabilities of most FPGAs. This function, which is used for debugging, allows a bitstream to be extracted from an FPGA device during run time. Of course, in a secure context, this feature has to be disabled, but this is not sufficient, since in low cost SRAM FPGAs, bitstream retrieval is very simple, even without a read back mechanism. Indeed the bitstream is stored in an external EEPROM, so the attacker can probe the data line between the FPGA and EEPROM to obtain it. This threat does not exist for non-volatile FPGAs because configuration data are stored inside the device, so only intrusive attacks are possible. Bitstream encryption mechanisms are available for most advanced FPGAs. A secret encryption key is stored inside the programmable device, while for volatile FPGAs an external battery is used to maintain key value. Thus the device accepts an encrypted bitstream and uses its dedicated decryption engine to obtain unciphered data. Attackers cannot decrypt the bitstream without the secret key. With this feature, attackers have to discover the secret key they need to recover bitstream data through (for example) an intrusive attack.

- *Attackers' objectives concerning bitstreams*—When an attacker discovers the FPGA configuration, a lot of information, some of which may be critical, becomes accessible. The first threat is bitstream reverse engineering; some companies are specialized in FPGA reverse engineering [14]. In this way, attackers could potentially access all information stored in the FPGA architecture, possibly secret cipher keys or maybe the structure of cryptographic algorithms not available to the public. Attackers can also inject their own bitstream into the programmable device, in this way reverse engineering becomes unnecessary, since the attacker could simply create a dummy system. For example, if the FPGA is used to encrypt written data to a hard disk drive, the attacker could build a system that does not cipher data. Another threat is fault injection into the bitstream [11]. Even without knowledge of the architecture, small modifications can seriously modify the system. Fault attacks are also suitable in this context. The final drawback of FPGA is cloning. An attacker in possession of the configuration data can clone the device ignoring possible intellectual property rights. This is not a real security weakness, rather an industrial threat that is specific to programmable devices. In conclusion, the bitstream is the image of the underlying FPGA architecture, it is similar to ASIC layout, therefore configuration data have to be protected.
- *Remote configuration*—Remote configuration is an interesting FPGA feature that allows systems to be upgraded by removing a potential security breach, or by upgrading algorithms. But this feature must be extremely well secured since it gives many possibilities to attackers. The first related threat is undesired reconfiguration, i.e., the design could be changed remotely by the attacker without user permission. A simple switch button could avoid this threat. The second is "man in the middle" attacks. The user wants to upgrade his/her programmable security device, but an attacker intercepts the request and replies with a fake configuration. For that reason, the connection must be secured by an authentication and integrity checking engine. Such secured mechanisms are already suitable for most FPGAs, Actel or Xilinx FPGAs, and application notes [2] describe secured remote configuration schemes.

Key management is also a major concern for embedded systems. This threat is not specific to FPGAs, ASICs are also vulnerable. Cryptographic devices need to store some symmetric and asymmetric keys to perform encryption or decryption. During system use, these cryptographic keys need to be changed or generated randomly. So these types of devices require secure key management.

2.6 Countermeasures

In the previous section we described some vulnerabilities of FPGAs from the technological level up to the system level. Fortunately it is possible to deal with these vulnerabilities to take advantages of the flexibility and the performances offered by FPGAs. In this section, we first review technological issues and then present logical solutions, such as dual-rail or asynchronous techniques. At the architecture level,

we highlight the fact that the parallelism and flexibility provided by FPGAs can be very efficient aids in building a secure primitive. Finally at the system level, the main issue is related to communication security and monitoring of the behavior of the system. We describe several techniques to address these problems.

2.6.1 Technological Level

At the technological level, the main countermeasures are related to physical processes, circuits, and packaging [49, 50]. Some sensors can be used to detect any attacks against the device. With standard FPGAs, the user is blocked by chip capabilities as most FPGAs do not have analog parts that can monitor environmental parameters. In the latest Actel FPGA, called Fusion, various analog functions are available like temperature sensors, clock frequency or voltage monitoring. These functionalities can be used to detect possible fault injection. To prevent physical attacks, it is also necessary to make sure that the retention effects of the cells are as small as possible, so that an attacker cannot detect the status of the cells. The solution would be to invert the data stored periodically or to move the data around in memory. Cryptographic applications cause long-term retention effects in SRAM memory cells by repeatedly feeding data through the same circuit. One example is specialized hardware that always uses the same circuits to feed the secret key to the arithmetic unit. This effect can be neutralized by applying an opposite current or by inserting dummy cycles in the circuit. In terms of FPGA application, this is very costly and it may even be impractical to provide solutions like inverting the bits or changing the location of the whole configuration file. One possible alternative would be to change only the crucial part of the design, like the secret keys, through dynamic partial reconfiguration. Counter techniques such as dummy cycles and opposite current approach can also be used for FPGA applications. Technological issues depend on the FPGA vendor but solutions like dynamic reconfiguration can provide some opportunities to strengthen the security of the device against certain tampering techniques that benefit from the retention effect of cells. Using sensors inside FPGAs will certainly become more common in the future to monitor the internal activity of FPGAs.

2.6.2 Logical Level

So far, the published countermeasures against attacks on FPGAs have mainly been inspired by techniques used to protect ASICs. The “Masking” and “Hiding” methods are among the most efficient ways of protecting FPGAs. Hiding consists in making the power consumption as constant as possible by using dual rail with precharge logic (DPL) and a four-phase protocol inspired by Asynchronous Logic. This ensures that:

- Transitions are independent of the target values (logical 0 or 1), because the information is encoded on two indiscernible wires;
- No leaks are caused by consecutive data correlation because of the intermediate precharge phase.

The most straightforward countermeasure built on the DPL is referred to as "wave dynamic differential logic" WDDL [43, 44]. The four-phase cadence is enforced by appropriate registers. In the meantime, the differential logic uses positive dual functions. The WDDL logic could be more secure in FPGAs than in ASICs, because the two dual gates are implemented in LUTs, which ideally are much alike. However some flaws have been underlined. Vulnerabilities are caused by the imbalance between the two networks and above all by the "early evaluation" (EE) effect. The latter is due to the differences in delay between two gate inputs. The effect is particularly marked in FPGAs where there is a wide range of routing. Associated with the need for speed and a low complexity design, new DPLs have been devised to enhance WDDL in FPGAs, among which: MDPL [35], STTL [36], BCDL [32].

The masking countermeasure allows the mean of the power consumption to be balanced by masking the sensitive data with a random variable [18, 45, 46]. Hence the observation is theoretically not exploitable by an adversary. In FPGA technology, the masked data are computed at the same time as the mask itself. This implementation, called "zero-offset" [47], allows the designer to keep a high level of ciphering throughput. However, whatever the implementation (hardware or software), the masking protection could give rise to higher order attacks [30]. Hence enhanced masking structures should be designed and used in FPGAs. One example is provided in the chapter on countermeasures.

Other countermeasures are possible at the logical level:

- *Random underlying operations*—FPGAs enable many more attacks to be defeated than the fixed architectures implemented in ASIC devices. A promising solution is to frequently modify the operation used in any particular cipher. For example, S-Box transformation used in the most recent symmetric bloc cipher (AES) could be computed in different ways, e.g. using Galois field $GF(2^n)$ operations, using a big lookup table or using the internal RAM memory embedded in most FPGAs. In this way, the power consumption would differ for each type of S-Box implementation.
- *Random noise addition*—A lot of countermeasures have been proposed, from clock randomization, power consumption randomization, or compensation [11] to tamper detection. However, longer differential power analysis generally leads to recovery of the secret key. With enough samples, the statistical analysis used by DPA can remove any random noise on power consumption. To understand why, we need to understand that DPA attacks succeed because a calculus always leaves the same power consumption trace. Adding random noise to power lines is not enough, because it could be removed by a number of different signal processing techniques. This can be represented by:

$$TotalPower(t) = RealPower(t) + RandomPower(t),$$

where *TotalPower* is the power trace that an attacker can easily measure, *RealPower* is the power consumed by the cryptographic algorithm that the attacker wishes to obtain, and *RandomPower* is the random noise added to defeat DPA. To extract the *RealPower*, the attacker could collect a large number of *TotalPower* traces and then average the collection. With enough *TotalPower* traces, the average will be very close to *RealPower*. According to this analysis, adding random noise is not a perfect solution, it could make DPA longer but would not make it impossible. A better way is to act directly on the *RealPower* factor.

In the FPGA context, other types of methods can be applied using runtime reconfiguration. For example, in the case of the AES algorithm, differential power analysis targets sub-byte operations. Power consumption is generally correlated with data involved in the sub-byte function because this operation is always done with the same logic gates. With partial reconfiguration, the sub-byte function could be done differently at every computation, so power consumption would be different. This method is similar to masking mechanisms, the input data and the final result may be the same but the underlying operations are not identical, so that power consumption is no longer correlated with the data. These different points will be analyzed in detail in the following chapters of this book.

2.6.3 Architecture Level

At the architectural level, some solutions use a method called LRA (leak resistant arithmetic) based on a random number representation [4, 7]. In these works, input data are represented according to a residue number system base which can be selected randomly before or during the encryption process. After computation, the result is converted into a classical binary representation. To help the reader understand this approach, we will explain some mathematical concepts. Leak Resistant Arithmetic (LRA) is based on the Residue Number System and on Montgomery's modular multiplication algorithm proposed in [4]. The Residue Number System (RNS) relies on the Chinese Remainder Theorem (CRT). This theorem states that it is possible to represent a large integer using a set of smaller integers. A residue number system is defined by a set of k integer constants, $m_1, m_2, m_3, \ldots, m_k$, called moduli. The moduli must all be co-prime; no modulus can be a factor of any other. Let M be the product of all the m_i. Any arbitrary integer X smaller than M can be represented in the residue number system as a set of k smaller integers $x_1, x_2, x_3, \ldots, x_k$ with

$$x_i = X \bmod m_i.$$

With this representation, simple arithmetic operations, like additions, subtractions and multiplications, can be performed absolutely in parallel. Once represented in RNS, the operations do not generate any carry. For example, adding A and B in RNS is just:

$$Sum_i = (a_i + b_i) \bmod m_i.$$

This sum involves k modular additions that can be performed in parallel. However, the conversion from RNS to decimals is not so simple. Fortunately, this conversion is only needed after all modular multiplications, so its cost is amortized. For cryptographic applications, modular reduction ($x \bmod M$), modular multiplication ($x * y \bmod M$) and modular exponentiation ($x^y \bmod M$) are some of the most important operations. They can be calculated using Montgomery's algorithm, modified for RNS representation. In addition to possible parallelism, LRA could also provide an algorithmic countermeasure against side channel attacks. With LRA, it is possible to compute the same calculus (RSA or ECC algorithms) in different bases. The goal of this approach is to randomize intermediate results. The same modular multiplication leads to different basic RNS operations, so power consumption will differ if the RNS base differs. RNS bases have to be chosen randomly so the attacker cannot obtain the base value, and by extension, the underlying RNS operations.

Another issue is related to fault injection at the architectural level with the aim of recovering sensitive data. A common way to avoid fault injection attacks is to use redundancy [2]. Critical parts in the design are replicated, and then outputs are compared, which generally allows errors to be detected. Mathematical error detection can also be used, [7] shows how to use redundant information to detect potential errors during calculus. This mechanism also relies on RNS, but an extra modulus is used to check the correctness of the result. Verification is possible at the end of each modular multiplication, so this type of attack can be detected. Moreover the physical location of each operator can be changed dynamically during the lifetime of the FPGA. Therefore, accurate fault injection using laser or focused ion beams is no longer possible, as attackers cannot identify the exact operator position.

Agility is another important metric for cryptographic applications since, as mentioned previously, providing a moving target increases the complexity of setting up an attack. To illustrate this point, the following case study deals with an AES security primitive. All selected implementations were performed on Xilinx Virtex FPGA, which is a fine grain configurable architecture. For this architecture, the configuration memory relies on a 1D configuration array. This is a column based configuration array and so partial configuration can be performed only column by column. For security issues, this type of configuration memory does not provide full flexibility but still enables partial dynamic configuration to perform security scenarios. Figure 2.9 summarizes all the different implementations in four charts; each chart corresponds to specific parameters. Figure 2.9.a corresponds to the AES cryptographic core security primitive with BRAMs (i.e. embedded RAM) on non-feedback mode [15, 20, 28, 37, 41]. Thus key setup management is not used for these studies. Concerning agility, all the solutions are based on static and full configuration. The configuration is defined through predefined configuration data and performed remotely. The configuration time is on average tens of ms, since full configuration is performed. The security module controller is not addressed in these studies since the implementations are static. Figure 2.9.a shows that various area/throughput trade-offs are possible depending on the implementation. It is important to dynamically adapt the performance and to ensure the security of the module. From the security point of view, it enables the global system to behave as a moving target, while from the

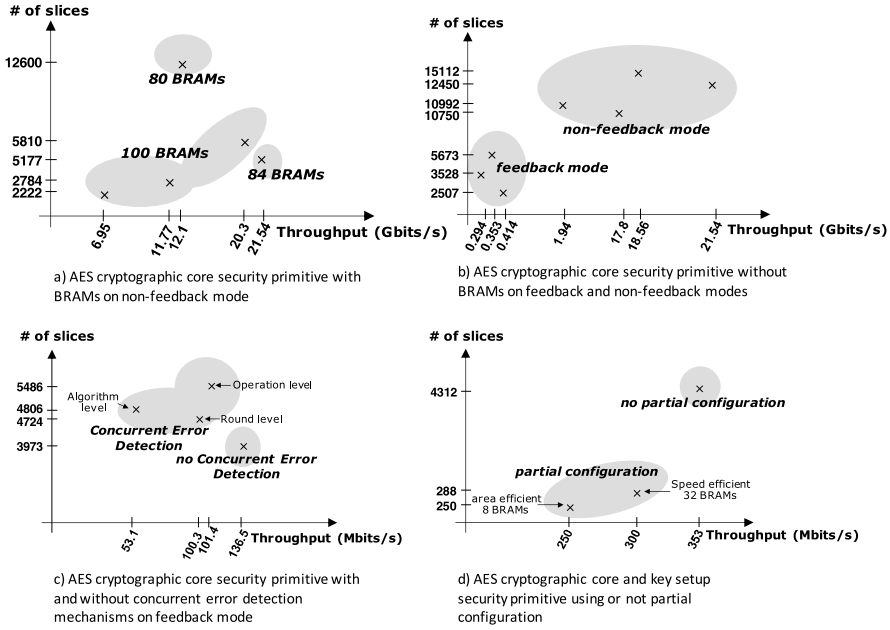


Fig. 2.9 Agility design space for the AES security primitive: throughput/area/reliability trade-offs

performance point of view, it allows different throughputs to be used dynamically depending on the actual requirements of the application. Figure 2.9.b corresponds to the AES cryptographic core security primitive without BRAMs on feedback [10, 13, 15] and non-feedback modes [13, 20, 22, 40]. Like in the previously example, key setup management is not used. Solutions [10, 13, 15] correspond to feedback mode while the others correspond to non-feedback mode. Feedback solutions enable throughput of on average hundreds of Mbits/s whereas non-feedback solutions enable around tens of Gbits/s. The same remarks as previously apply to agility characteristics; static, full and predefined configuration is used. In these studies the goal is to promote high throughput while reducing area and dealing with a specific execution mode. However as explained in this section, dynamism and reliability also have to be considered.

Figure 2.9.c shows different ways to manage fault detection that ensure reliability, an essential feature for security. Faults can be detected at different levels of granularity from algorithm to operation level [23]. The performance/reliability trade-off is interesting since a finer level of granularity enables reduced fault detection latency and facilitates a rapid reaction against an attack. But the price of this efficiency is area overhead. No type of error detection can improve performance, it is thus important to dynamically adapt the level of protection depending on the environment and on the state of the system. Concerning agility, static, full and predefined configuration is used. Finally, Fig. 2.9.d provides some interesting values since solutions using dynamic configuration are proposed. In [10], full configuration with predefined configuration data is implemented, whereas in [29] partial configuration

with dynamic configuration data is performed. In both cases, remote configuration is performed since the Configurable Security Module is considered be an agile hardware accelerator. Both solutions also deal with key setup management, in [10] this is performed inside the module so that the architecture is generic and in [29] it is performed by the remote processor, which means key specific architecture can be provided. In [29] the remote processor implements the security module controller that computes the new configuration when new keys have to be taken into account by the cryptography core. This type of execution enables considerable flexibility since the configuration data can be defined at run time. However in that case, the computation time to define the new configuration data is in the range of 63–153 ms, which may be prohibitive for some applications. The reconfiguration time for new configuration data is not critical (around tens of μs) since only partial configuration is performed. As can be seen in Fig. 2.9.d, partial configuration enables significant area savings compared to a generic implementation since in the latter, the architecture is specialized for each key. The security policy supported by the security module controllers are not explicitly presented in these works. Figure 2.9 shows that different solutions can be implemented for the same security primitive and so different area/throughput/reliability trade-offs can be considered. Agility enables these trade-offs and consequently both performance and security to be dynamically adapted to the actual execution context. The last important point is related to power consumption which has not been tackled in previous studies, even though for embedded systems, it may be an essential feature. In [38], energy efficient solutions are proposed for the AES security primitive. In this case, the important metric is Gbits/joule, which is very relevant since ambient systems are mobile.

It is important for designers who have to build modules to be aware of all these trade-offs in order to promote agility and to meet performance requirements. Further detailed studies on configuration power consumption, secure communication links and security module controller policy are required in order to propose secure modules and by extension, secure systems. However agility provides many keys to building high-security/high-performance systems.

2.6.4 System Level

Generating a perfectly secured system is an illusion; so the best way to prevent unanswerable threats is to detect them and to respond appropriately. Attacks against secured devices vary and many parameters have to be monitored to detect them. The SAFES (Security Architecture for Embedded Systems) approach focuses on protecting embedded systems by providing an architectural support for the prevention, detection and remediation of attacks [17]. Most embedded systems are implemented as System on a Chip devices, where all important system components (processor, memory, I/O) are implemented on a single chip. This solution extends the functionality of such systems to include both reconfigurable hardware and a continuous monitoring system that guarantees secure operations. Through monitoring, abnormal behavior of the system can be detected and hardware defense mechanisms can

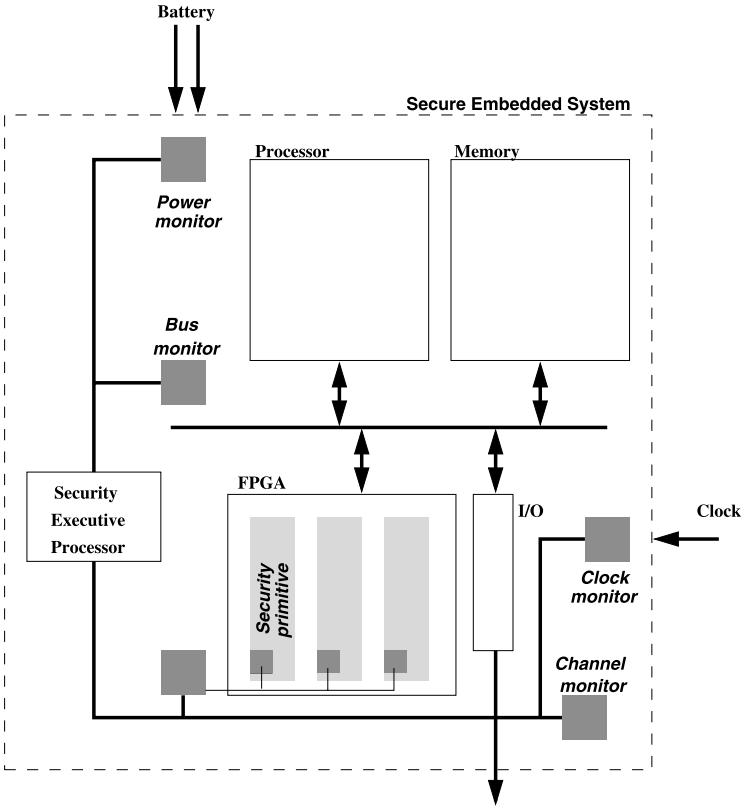


Fig. 2.10 The Security Architecture for Embedded Systems. The reconfigurable architecture contains the security primitives and the monitors protect the system

be used to fend off attacks. Such an approach has several advantages since application verification and protection is provided by dedicated hardware and not inside the application. The security mechanisms can be updated dynamically depending on the application running on the system, which guarantees the durability of the architecture. The SAFES approach focuses on embedded security and exploits the characteristics of embedded computations.

Figure 2.10 provides an overview of the architecture. As we can see, several monitors are used to track system-specific data. The number and complexity of the monitors are important parameters as they are directly related to the overhead cost of the security architecture. The role of these monitors is to detect attacks against the system. To provide such a solution, the normal activity (i.e. correct or expected) behavior of the modules is characterized in such a way that irregular behavior is detected. Autonomy and adaptability have been stressed to build an efficient security network of monitors. The monitors are autonomous in order to build fault tolerant systems; if one monitor is attacked, the others continue to manage the security of the

system. The monitors are distributed so as to be able to analyze the different parts of the system (e.g. battery, buses, security primitives, communication channels).

Different levels of reaction, reflex or global, are used depending on the type of attack. A reflex reaction is performed by a single monitor, in this case the response time is very short since no communication is required between the different monitors. A global reaction is performed when an attack involves a major modification of the system. In this case, the monitors need to define a new global configuration of the system, which requires a longer response time. The monitors are linked by an on-chip intelligence network. This network is controlled by the Security Executive Processor (SEP) that acts as a secure gateway to the outside world. The SEP provides a software layer to map new monitoring and verification algorithms to monitors. This point is important to meet FIPS 140-2 requirements, which require that data I/Os and security I/Os are implemented through separate links. In response to abnormal behavior, the SEP can issue commands to control the operation of the system. For example, it can override the power management or disable I/O operations.

Detecting an attack is a good thing, but a good response to a threat is of course needed. In order to qualify for FIPS 140-2 security level 2, the secret information has to be deleted in a short time. In an FPGA context this could be a problem if cipher keys are stored in the bitstream, because deleting bitstream memory could take too long for FIPS recommendations. For FIPS 140-2 levels 3 and 4, the task is even harder, as the entire secured device has to be destroyed. Such devices are rare and are reserved for government use, for example for nuclear weapons. A common reaction is to trigger explosives if an attack is detected, but chemical products can be used instead. These mechanisms could be used in the FPGA context, but another solution is possible. If the configuration memory and the FPGA content are erased, we can consider that the secured device is destroyed. Thus the job is dual-purpose, the FPGA content has to be destroyed, and the configuration memory has to be erased because it is the image of the underlying FPGA content. For SRAM FPGA devices, the first problem could be resolved by removing the FPGA power supply, which would result in the removal of the volatile configuration. In secured volatile FPGAs, erasing the bitstream memory can be avoided. In fact these devices embed a secret key used to decrypt the bitstream, and configuring data decryption is no longer possible when this key is erased. The need to erase configuration memory could be replaced by removing the bitstream secret key. As volatile FPGAs require a battery to maintain the configuration of the secret key, secured volatile FPGA devices could be considered destroyed if all power sources are removed (secure configuration battery and FPGA power supply). For non-volatile technologies, such a simple mechanism cannot be used because the secret key and bitstream data are not erased when the power is off. Of course configuration memory is not a problem for non-volatile FPGAs.

To perform a non-invasive attack such as DPA or DEMA, attackers need to collect a lot of leakage information computed with the same cipher key. The system could detect an abnormal use of a device by fixing a maximum number of operations allowed with the same key, for example. As previously mentioned, fault injection

could be detected at lower levels. But redundancy in attack detection is not a bad thing as it allows different types of attacks to be countered. With standard FPGAs, the user is limited by chip capabilities, most FPGAs do not have analog parts that can monitor environmental parameters. With the latest Actel FPGA, Fusion, various analog functions are available including temperature sensors, clock frequency or voltage monitoring. These functionalities can be used to detect possible fault injection.

In order to overcome standard FPGA deficiencies (no analog parts for example), one solution is to use tamper proof sealing, like IBM PCI Cryptographic Coprocessor product [21]. Invasive attacks are no longer possible, and the metal shield also protects against electromagnetic analysis attacks. With such a mechanism, many parts like analog sensors could be placed in a trusted environment near the FPGA, thereby extending the capabilities of the programmable device. This is not possible without tamper proof sealing because an attacker could freely act on analog sensors or even remove them.

One major concern with FPGAs at the system level concerns communications between the different resources within the system. Depending on the trusted area coverage within the system, it is essential to protect the communications and to minimize the overhead due to cryptography primitives. Some solutions are presented below. Their goal is to lessen the cost of confidentiality and integrity.

PE-ICE is a dedicated solution providing strong encryption and integrity checking to data transferred on the processor-memory bus of an embedded computing system [12]. It was designed to optimize latencies introduced by the hardware security mechanisms on read and write operations. To achieve this goal, PE-ICE uses a single block encryption algorithm. Data confidentiality is thus guaranteed by encryption while integrity checking relies on the spreading feature of block encryption algorithms and on resources available on chip.

Data privacy is provided by AES (Advanced Encryption Standard) encryption. AES is a block cipher algorithm that processes 128-bit blocks with a 128-bit key.

The spreading feature of block encryption algorithms implies that once a block encryption is performed, the position and the value of all bits in a ciphertext block C are influenced by each bit of the corresponding plaintext block P. Consequently if P is composed of two distinct data (PL and T), after ciphering, it is impossible to distinguish the PL ciphered part from the T part in C. Moreover if one bit is modified in C, after decryption all bits of the corresponding plaintext block will be affected.

PE-ICE integrity checking process: The previous property is used to add the integrity checking capability to block encryption. In a write operation (Fig. 2.11), a tag value is inserted in each plaintext block P before encryption. As a result, P is composed of a payload PL (data to protect) and a tag (T). Such a tag must be a nonce, a Number used ONCE, for a given encryption key and does not need to be calculated over the data with a specific algorithm; for example, it can be generated by a counter. After encryption, an indistinguishable and unique PL/T pair is created and the resulting ciphered block C is written in the external memory. In a read operation (Fig. 2.11) C is loaded and decrypted. The tag T derived from the resulting plaintext block is compared to an on chip regenerated tag called the reference tag T. If

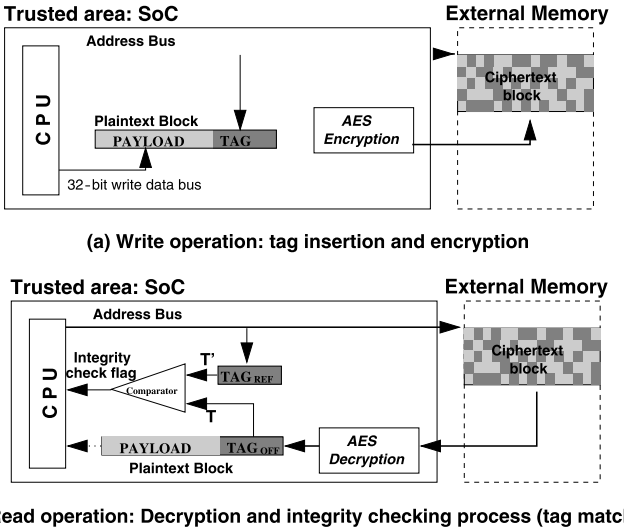


Fig. 2.11 PE-ICE principle: the protection is based on the spreading feature on block encryption algorithms

T does not match T, it means that at least one bit of C has been modified (spoofing attack), PE-ICE raises an integrity checking flag to prevent further processing.

The tag generation: In the context of processor—memory communication, the hardware engine executes both encryption and decryption. Therefore, the engine has to hold the tag value T of each ciphered block between encryption and the decryption or alternatively, must be able to regenerate it in read operations to perform the integrity checking process. The challenge is to reach this objective by storing as little tag information as possible on the engine to optimize on chip memory usage. Moreover T may be public knowledge because an adversary needs the secret encryption key to create an accepted PL/T pair. CPUs process two kinds of data, Read Only (RO) data and Read/Write data (RW). The composition of the tag differs for each kind of data and depends on their respective properties. RO data are only written once in external memory and are not modified during program execution. In addition, the secret encryption key is changed for each application downloaded in the external memory. Therefore, the tag can be fixed for each plaintext block of RO data. PE-ICE uses the most significant bits of the ciphered block address as tags (Fig. 2.11.a). If an attacker launches a splicing attack, the address used by the processor to fetch a block and by PE-ICE to generate the tag reference (T) will not match the one loaded as the tag (T). RW data are modified during software execution and are consequently sensitive to replay attacks. Using only the address as the tag is not enough to prevent such attacks because the processor cannot check if the data stored at a given address is the most recent data (temporal permutation). For that reason, the tag is composed of the most significant bits of the address of each ciphered block, concatenated with a random value (Fig. 2.11.b). The random value

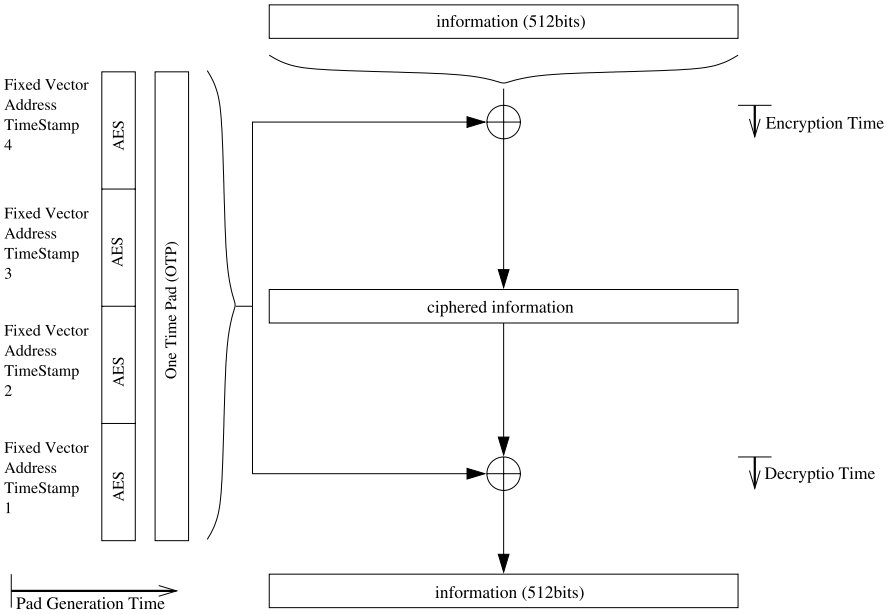


Fig. 2.12 One Time Pad model: ciphering and deciphering are performed through a XOR operation

of each plaintext block is changed for each write operation and is stored on chip to be able to regenerate the tag reference during read operations.

The OTP (One Time Pad) approach is an alternative solution to PE-ICE that tries to further parallelize the decryption time and the fetch latency. If we consider a block of 512 bits, i.e. the same size as the cache line that corresponds to a keystream and that is created using an AES algorithm, this block is created with the address of the data, a timestamp and a fixed vector (these data are required to protect the memory against classical attacks, spoofing, relocation, and timing). Except for the timestamp, all the information is known by the processor before memory access. Considering the possibility of using a special (fast) cache for timestamp storage, while the processor is fetching the data, a hardware engine computes the keystream (required for OTP). When the data are ready, the computation to decipher the information is only a bit-a-bit XOR gate (principle of OTP). Figure 2.12 shows the general model and the benefit of using OTP compared to classical ciphering techniques. If the memory access time is longer than the AES computation time, the deciphering latency is completed hidden by the model, as can be seen in Fig. 2.13.

Code compression techniques can also be used to reduce the cost of cryptography [48]. This solution aims to overcome the main problem of memory encryption i.e. the encryption delay. In general the decryption unit is defined in the secure area of the system, between the processor cache and the main memory, so that, by observing the bus activity, an attacker will be able to find encrypted information. This approach will impose an overhead in the processing time due to the deciphering unit.

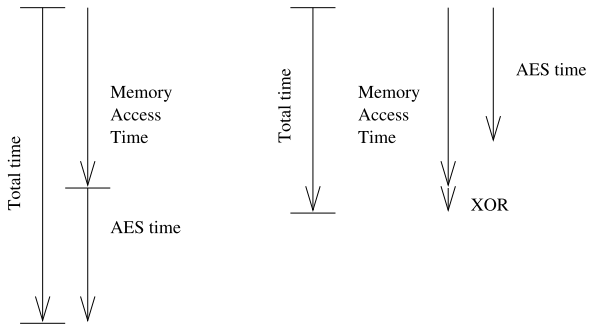


Fig. 2.13 OTP access time: keystream generation time is hidden by the memory access time. Ciphering is performed through a XOR operation

Using code compression, the number of transfers between the cache and the main memory will be reduced, which means that the global decryption overhead will be minimized, and if a scheme that naturally improves performance is used, the overhead can be completely avoided.

The general idea is presented in Fig. 2.14. After the compression phase, the AES algorithm is used to encrypt the blocks of 4×32 bits (the same size as a cache line). The compressed and encrypted code is then loaded into the main memory. When the processor asks for an instruction, the decompressor asks the cache if it is available. In this case, no deciphering is necessary. If the instruction in the cache is a codeword, the decompressor acts by delivering the corresponding instructions. If the instruction

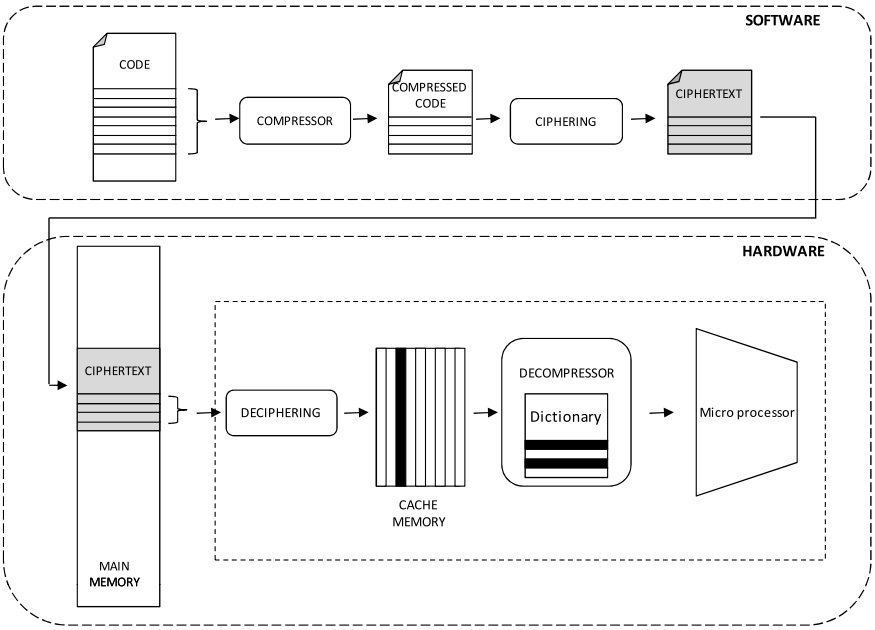


Fig. 2.14 Code compression techniques to mitigate the cost of cryptography

is not present in the cache, the deciphering unit is used, retrieving the block from the main memory and converting it into a compressed code that is delivered to the cache. This method relies on the efficiency of the compression to avoid some cache misses and, consequently accesses to the main memory. Moreover, the density of the code that is transferred through the bus is higher, so that less deciphering is required for the whole execution of the code.

All the solutions presented above allow for a more efficient and secure design at the system level. These solutions could also be considered for ASICs and are not fully dedicated to FPGAs. However, the advantages of FPGAs should not be considered as dealing only with a single level but with the whole security pyramid. In such a case, the different contributions e.g. dynamic reconfiguration at all the levels (i.e. technological, logic, architecture and system) and logic security improvements provide some very interesting features and should encourage the use of FPGAs for secure embedded systems.

2.7 Conclusions

Embedded systems are currently facing an increasing number of attacks since the amount of digital private information embedded in these systems is getting bigger every day. The rate of digital data exchanged is also increasing exponentially and transferred information must be kept secure since confidentiality and integrity are mandatory. Embedded systems are playing an essential role in our society and are already included in many electronic devices from low end to high end systems. Security is a serious problem and attacks against these systems are becoming more critical and sophisticated. New solutions are needed to allow the definition of secure embedded systems since current technologies are facing several challenges and cannot cope with security requirements and performance constraints. Architectures will have to meet high performance requirements, be energy efficient, flexible, tamper resistant and reliable to enable their wide adoption. FPGAs can address these requirements and provide efficient security primitives. Their characteristics enable the system to prevent attacks or to react when attacks are detected while guaranteeing the required energy and computation efficiency.

In this chapter we have analyzed current threats against embedded systems and especially FPGAs. We have described the standard FIPS requirements to build a secure system. This point is of paramount importance as it guarantees the level of security of a system. We have also highlighted current vulnerabilities of FPGAs at all the levels of the security pyramid. From a design point of view, it is essential to be aware of all these levels in order to provide a comprehensive solution. As mentioned earlier in this chapter, the strength of a system is defined by its weakest point; there is no reason to enhance other means of protection, if the weakest point is not tackled. Many serious attacks target this weakness in order to avoid facing the complexity of brute force attacks. Several solutions have been outlined in this chapter especially at the logical, architectural, and system levels to find a global solution.

References

1. Abraham, D.G., Dolan, G.M., Double, G.P., Stevens, J.V.: Transaction security system. IBM Syst. J. (1991)
2. ACTEL: Proasic3/e security. In: Application Note. ACTEL Corporation (2005)
3. Agrawal, D., Rohatgi, P., Rao, J.R.: Multi-channel Attacks. Lecture Notes in Computer Science (2003)
4. Bajard, J.-C., Imbert, L.: Leak resistant arithmetic. In: Cryptographic Hardware and Embedded Systems, Proceedings of CHES (2004)
5. Barker, E., Roginsky, A.: NIST special publication 800-131a—transitions: recommendation for transitioning the use of cryptographic algorithms and key lengths. In: Computer Security Division—Information Technology Laboratory (January 2011)
6. Barthe, L., Benoit, P., Torres, L.: Investigation of a masking countermeasure against side-channel attacks for RISC-based processor architectures. In: FPL, pp. 139–144 (2010)
7. Ciet, M., Nevel, M., Peeters, E., Quisquater, J.-J.: Parallel FPGA implementation of RSA with residue number systems—can side-channel threats be avoided. In: UCL Crypto Group (2004)
8. Coburn, J., Ravi, S., Raghunathan, A., Chakradhar, S.: SECA security-enhanced communication architecture. In: Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (2005)
9. Dagon, D., Martin, T., Starner, T.: Mobile phones as computing devices: the viruses are coming! IEEE Pervasive Comput. **3**(4) 11–15 (2004)
10. Dandalis, A., Prasanna, V.K.: An adaptive cryptographic engine for Internet protocol security architectures. ACM Trans. Des. Autom. Electron. Syst. **9**, 333–353 (2004)
11. Daniel, M., Techer, J.-D., Torres, L., Robert, M., Cathebras, G., Sassatelli, G., Moraes, F.: Current mask generation: an analogical circuit to thwart DPA attacks. In: IFIP International Federation for Information Processing (2006)
12. Elbaz, R., Champagne, D., Gebotys, C., Lee, R., Potlapally, N., Torres, L.: Hardware mechanisms for memory authentication: a survey of existing techniques and engines. In: Transactions on Computational Science IV. Lecture Notes in Computer Science, vol. 5430. Springer, Berlin (2009)
13. Elbirt, A.J., Yip, W., Chetwynd, B., Paar, C.: An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists. IEEE Trans. Very Large Scale Integr. **9**, 545–557 (2001)
14. FPGA: Reverse engineering services. http://www.bltinc.com/services.fpga-reverse-engineering.htm
15. Gaj, K., Chodowiec, P.: Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays. In: Proc. RSA Security Conf. (2001)
16. Giraud, C.: DFA on AES. Technical report, Oberthur Card Systems (2004)
17. Gogniat, G., Wolf, T., Burleson, W., Diguet, J.-P., Bossuet, L., Vaslin, R.: Reconfigurable hardware for high-security/high-performance embedded systems: the safes perspective. IEEE Trans. Very Large Scale Integr. **16**(2) 144–155 (2008)
18. Gueron, Sh., Parzanchevsky, O., Zuk, O.: Masked inversion in GF($2n$) using mixed field representations and its efficient implementation for AES. In: Smart Card System Engineering, System LSI Division, Device Solutions Network. Samsung Electronics Co. Ltd (2006)
19. Guilley, S., Pacalet, R.: SoC security: a war against side-channels. Annals of the Telecommunications. Système sur puce Électronique pour les Télécommunications (2004)
20. Hodjat, A., Verbauwhede, I.: A 21.54 Gbits/s fully pipelined AES processor on FPGA. In: Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines (2004)
21. IBM: IBM PCI cryptographic coprocessor. In: General Information Manual. IBM Corporation (2002)
22. Järvinen, K.U., Tommiska, M.T., Skyttä, J.O.: A fully pipelined memoryless 17.8 Gbps AES-128 encryptor. In: Proceedings of the ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays (2003)

23. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2002)
24. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. Lecture Notes in Computer Science (1996)
25. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. Lecture Notes in Computer Science (1999)
26. Liardet, P.-Y., Teglia, Y.: Fault resistance: from reliability to safety. In: Proceedings of the International Conference on Dependable Systems and Networks (2004)
27. Martin, T., Hsiao, M., Ha, D., Krishnaswami, J.: Denial-of-service attacks on battery-powered mobile computers. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (2004)
28. Mcloone, M., Mccanny, J.V.: High performance single-chip FPGA Rijndael algorithm implementations. In: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (2001)
29. Mcmillan, S., Cameron, C.: JBITS implementation of the advanced encryption standard (Rijndael). In: Proceedings of the International Conference on Field-Programmable Logic and Its Applications (2001)
30. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: CHES, Worcester, MA, USA, August 17–18. LNCS, vol. 1965, pp. 71–77. Springer, Berlin (2000)
31. Nash, D., Martin, T., Ha, D., Hsiao, M.: Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops (2005)
32. Nassar, M., Bhasin, S., Danger, J.-L., Duc, G., Guilley, S.: BCDL: a high performance balanced DPL with global precharge and without early-evaluation. In: DATE'10, Dresden, Germany, March 8–12, 2010, pp. 849–854. IEEE Comput. Soc., Los Alamitos (2010)
33. National Institute of Standards Technology: Security requirements for cryptographic modules (FIPS pub 140-2). In: Federal Information Processing Standards Publication. National Institute of Standards and Technology (NIST) (2001)
34. Örs, S.B., Oswald, E., Preneel, B.: Power-analysis attacks on an FPGA—first experimental results. In: Proceedings of the CHES 2003 (2003)
35. Popp, T., Mangard, S.: Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In: Proceedings of CHES'05, Edinburgh, Scotland, UK, September 2005. Lecture Notes in Computer Science, vol. 3659, pp. 172–186. Springer, Berlin (2005)
36. Razafindraibe, A., Robert, M., Maurine, P.: Analysis and improvement of dual rail logic as a countermeasure against DPA. In: PATMOS, Göteborg, Sweden, pp. 340–351 (2007)
37. Saggese, G.P., Mazzeo, A., Mazzocca, N., Strollo, A.G.M.: An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In: Proceedings of the International Conference on Field-Programmable Logic and Its Applications (FPL 2003) (2003)
38. Schaumont, P., Verbauwhede, I.: Domain specific tools and methods for application in security processor design. Des. Autom. Embed. Syst. **7**, 365–383 (2002)
39. Standaert, F.-X., Örs, S.B., Preneel, B.: Power analysis of an FPGA: implementation of Rijndael: is pipelining a DPA countermeasure. In: Proceedings of the CHES 2004 (2004)
40. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D.: Efficient implementation of Rijndael encryption in reconfigurable hardware: improvements and design tradeoffs. In: Proceedings of Cryptographic Hardware and Embedded Systems (2003)
41. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J., Legat, J.-D.: A methodology to implement block ciphers in reconfigurable hardware and its application to fast and compact AES Rijndael. In: Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays (FPGA 2003) (2003)
42. Standaert, F.-X., Örs, S.B., Quisquater, J.-J., Preneel, B.: Power analysis attacks against FPGA implementations of the DES In: Proceedings of the FPL 2004 (2004)
43. Tiri, K., Verbauwhede, I.: Synthesis of secure FPGA implementations. In: Proceedings of International Workshop on Logic and Synthesis (IWLS 2004) (2004)

44. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: Design Automation and Test in Europe Conference (DATE 2004) (2004)
45. Trichina, E.: Combinational logic design for AES subbyte transformation on masked data. In: Smart Card System Engineering, System LSI Division, Device Solutions Network. Samsung Electronics Co. Ltd (2006)
46. Trichina, E., Korkishko, T.: Secure AES hardware module for resource constrained devices. In: Smart Card System Engineering, System LSI Division, Device Solutions Network. Samsung Electronics Co. Ltd (2006)
47. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: CHES. Lecture Notes in Computer Science, vol. 3156, pp. 1–15. Springer, Berlin (2004)
48. Wanderley, E., Vaslin, R., Gogniat, G., Diguet, J.-P.: A code compression method to cope with security hardware overheads. In: 19th IEEE International Symposium on Computer Architecture and High Performance Computing (2007)
49. Wollinger, T., Paar, C.: How secure are FPGAs in cryptographic applications. In: Proceedings of the 13th International Conference on Field-Programmable Logic and Applications (2003)
50. Wollinger, T., Paar, C.: Security aspects of FPGAs in cryptographic applications. In: Lysaght, P., Rosenstiel, W. (eds.) New Algorithms, Architectures and Applications for Reconfigurable Computing, pp. 265–278. Springer, Dordrecht (2005)

Chapter 3
Side Channel Attacks

V. Lomné, A. Dehaboui, P. Maurine, L. Torres, and M. Robert

Abstract This chapter presents the main Side-Channel Attacks, a kind of hardware cryptanalytic techniques which exploits the physical behavior of an IC to extract secrets implied in cryptographic operations. We show in this chapter the main modern concepts about Side Channel Attacks (Simple and Differential Power Analysis) and how they can be deployed on FPGA architecture. We give also a set of details on platform and equipment needed to conduct such type of experiments. Then we propose a discussion about the leakage model of digital IC, comprising FPGA, and we illustrate these attacks on a set of real case study. We conclude this chapter by giving the latest information and link toward new efficient Side Channel Attacks.

3.1 Introduction

In the past 100 years, we have seen the emergence of modern cryptography, along with many cryptographic primitives and protocols. The development of new theoretical cryptanalytic techniques to try to defeat the main cryptographic algorithms has increased knowledge of how to design cryptographic primitives and schemes.

In theoretical cryptanalysis today, a cryptographic algorithm is considered as a black box. Even when attackers know the cryptographic algorithm, they only have access to pairs of plaintexts/ciphertexts, and their goal is to guess the cryptographic key.

The robustness of modern cryptographic algorithms is based on these assumptions, and a cryptanalytic attack is a method that allows an assailant to guess the key with a complexity (in time and/or in memory) lower than a brute force attack.

But if a cryptographic algorithm is modeled as a gray box, i.e. in such a way that an attacker can obtain intermediate information during the cryptographic operation requiring the key, these assumptions no longer hold true.

In this context, and with the increasing use of embedded cryptographic devices with embedded cryptographic secrets, several tamper attacks have appeared since

L. Torres (✉)
LIRMM—UMR CNRS 5506, University of Montpellier 2, Montpellier, France
e-mail: lionel.torres@lirmm.fr

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_3, © Springer Science+Business Media B.V. 2011

the beginning of the 1990s. These enable an attacker to obtain intermediate information during the cryptographic operation, and then to deduce the secret key with a complexity much lower than classical theoretic cryptanalytic attacks.

Figure 3.1 shows different models from theoretical and hardware cryptanalytic points of view. In the gray box model, physical leakages from the device can be identified or the computation running on the device can be disturbed. As we explain in this chapter, these different approaches enable cryptographic secrets to be extracted. Cryptanalytic techniques for hardware are classified as invasive attacks, semi-invasive attacks, or non-invasive attacks. We describe these attacks in detail in the following sections.

3.1.1 Invasive Attacks

Invasive attacks are a tamper attack in which the device is completely destroyed in order to extract secret information. The best known invasive attack is hardware reverse engineering, as described in [69]. The goal is to retrieve the layout of the circuit using chemistry techniques and/or high resolution microscopy. This technique is often used for cloning or anti-cloning purposes.

The first step consists in decapsulating the chip. A high resolution picture is then taken of each metal layer, and chemistry techniques are used to remove each succeeding metal layer to penetrate deeper into the chip right down to the transistor level. When attackers have a picture of each metal layer, they can use dedicated tools to go back to the netlist of the circuit. Finally, they obtain the behavioral description of the chip.

This attack has been conducted in the case of the MyFare stream encryption algorithm, which was kept secret by NXP. An invasive attack revealed the algorithm's functionality [46]. The method is shown in Fig. 3.2.

Although this technique is very powerful, but it has two drawbacks. Firstly, it requires very expensive equipment and highly qualified engineers. Secondly, thanks to technology shrinking techniques, it requires more and more sophisticated microscopy. In the case of SRAM and Flash based FPGAs, the configuration file, also called the bitstream, is stored in a dedicated off or on chip Flash memory. Thus, considering that the layout of an FPGA is roughly an array of reconfigurable cells, invasive attacks that allow attackers to obtain the hardware description of the attacked chip still provide no information about the configuration file, because the reconfigurable cells lose their configuration data as soon as the FPGA is switched off.

For instance, Fig. 3.3 shows the layout of an FPGA; as can be seen, optical observation of the floorplan provides no clues about the FPGA's configuration.

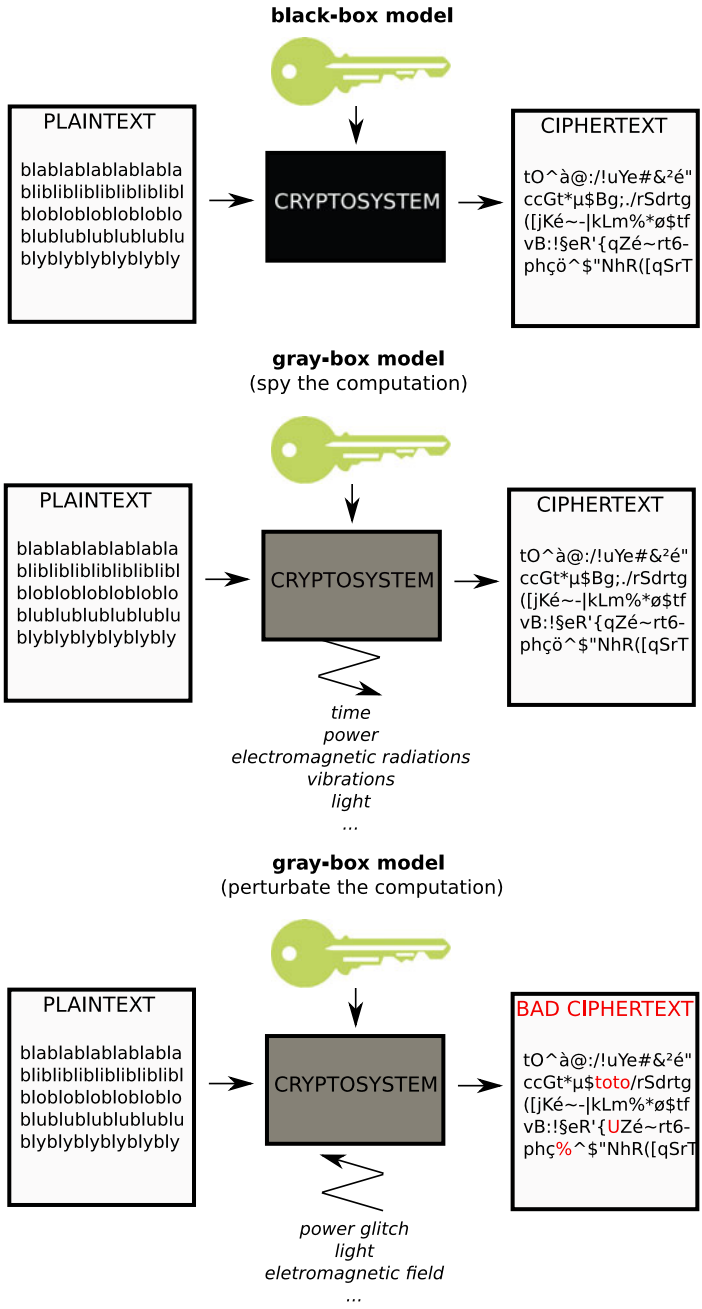


Fig. 3.1 Black-box and gray-box models from a cryptanalytic point of view

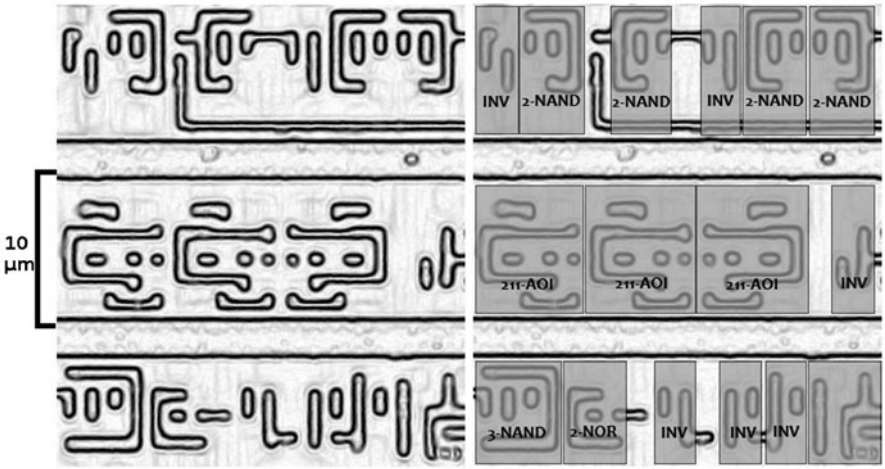


Fig. 3.2 Reverse-engineering of the MyFare CRYPTO1 algorithm. *On the left*, pictures of the circuit. *On the right*, netlist reconstruction

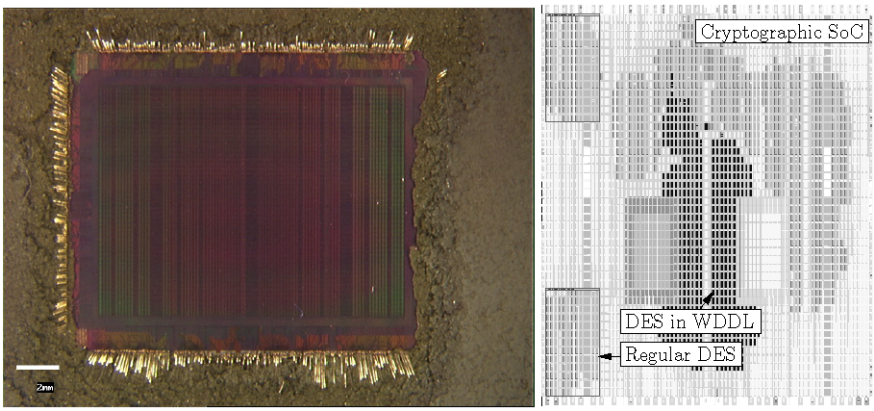


Fig. 3.3 Layout of a Stratix FPGA (ALTERA). *On the left*, a magnified photograph of the silicon die. *On the right*, the application actually programmed in the matrix, viewed under QUARTUS but invisible under the microscope

3.1.2 Semi-invasive Attacks

Semi-invasive attacks consist in retrieving physical information or disturbing the cryptographic operation computed by the device after decapsulation. These kinds of attacks are called semi-invasive because they require physical modification of the device, but do not destroy it. Indeed, attackers usually uses chemistry techniques to remove the package of the chip. In some cases, other sophisticated techniques may

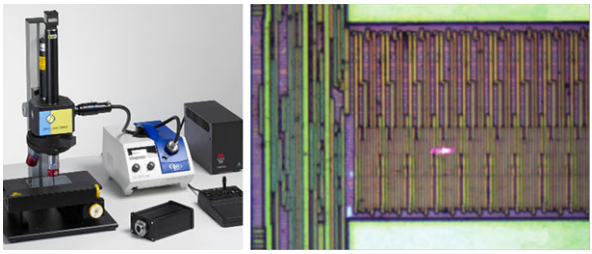


Fig. 3.4 Commercial laser attack setup from Riscure (*on the left*)—zoomed view of memory with the laser pulse in *pink* (*on the right*) (color online)

be required for example thinning down the substrate. For this purpose, different approaches can be used.

The first approach belongs to the *fault attacks* category. One or several faults are generated during the cryptographic operation using, for instance, light pulses. A concrete light based fault attack was first reported in [63]. Since these first concrete results, more sophisticated techniques have been elaborated using a laser (for instance red or infra-red diode laser), allowing the fault to be generated on either the front or back of the chip, and/or the fault to be focused on one or a few bits [8]. Figure 3.4 shows a commercial laser attack setup from Riscure (on the left), and zoomed view of memory with the tiny laser pulse in pink (on the right).

Another technique, called *probing attack* [32], consists in placing a small needle on a bus line of the microcontroller to retrieve information. Different authors have shown that even if attackers can only spy on one random bus line, they can nevertheless obtain enough information to reveal a cryptographic key. As an example, the authors of [62] describe a probing attack applied on AES. Recently, a general side-channel method based on single bit probing [24] was introduced as a way to optimize cube attacks [25]. Such attacks use a learning step, and can consequently even work on a proprietary algorithm.

A recent kind of semi-invasive attack consists in measuring, on the back side, photons emitted by transistors when they switch. This attack [13] requires thinning down the substrate and a high performance camera. Then, after the acquisition of photons emitted by the chip during the cryptographic computation of different data and the same unknown key, a statistical treatment, similar to that used in SCA reveals the cryptographic key. Moreover, this powerful technique follow the displacement of the current to be monitored by computing a movie disclosing the switching of transistors. However, this attack requires very expensive equipment that is available only in a few advanced laboratories.

3.1.3 Non-invasive Attacks

Non-invasive attacks are powerful techniques to extract secret information from cryptographic devices without physical modification of the device. As explained above, we distinguish two approaches, one consists in identifying physical leakages during the cryptographic operation (called *Side Channel Attacks* (SCA)), the other in disturbing computation (called *Fault Attacks* (FA)).

Side Channel Attacks consist in measuring physical leakages from the device during a cryptographic operation. The main idea behind this approach is that CMOS technology has an interesting property from a cryptanalytic point of view: it leaks physical information correlated with processed data. This class of attacks is very powerful, and requires only affordable measuring and testing equipment. Different physical leakages can be measured. Typical leaks are the computational time of the cryptographic operation [35], power consumption by the device [36], or the electromagnetic radiations emitted by the chip [26].

In some cases secret information can be extracted via a physical leakage from only one cryptographic operation, in others the attacker needs to record physical leakages from several cryptographic operations and to apply a statistical treatment on these records. On a classical smart card without dedicated countermeasures, several thousand records of cryptographic operations enable both symmetric ciphers (like DES or AES) and asymmetric ciphers (like RSA or ECC) to be broken in a very short computational time (a few hours using a desktop computer).

The first type of *Side-Channel Attacks* is the *Timing Attack*, which exploits differences in the computational time of a cryptographic operation. But from a practical point of view, methods have been proposed to implement a cryptographic algorithm in both hardware or software, ensuring that its computational time is constant.

However, power and electromagnetic side channels are physical leakages that are tightly correlated with the processed data due to CMOS properties, and these kinds of attacks are considered to be very powerful by manufacturers and government agencies. Moreover, using a tiny electric or magnetic sensor that is smaller than the chip, a 3D electromagnetic radiation map can be computed. This is achieved by running the same set of instructions several times, while placing the sensor in different positions.

The second approach belongs to the *fault attacks* category. The initial idea, from [17], is to run the cryptographic operation twice, one safe and another faulty. Dedicated cryptanalytic techniques enable one pair of safe/faulty cryptographic computations to be exploited to extract the key, with a low complexity in comparison to classical cryptanalytic techniques. In other cases a differential approach is required to retrieve the key [16], in which case the attacker needs several pairs of safe/faulty cryptographic computations to retrieve the cryptographic key.

The first concrete way proposed in the literature to generate faults during a computation is the *glitch attack* [10]. The idea is to modify the supplying signal of the device during a sharp time shorter than a clock period. The device can be overpowered or underpowered. As a result, a setup time or hold time violation will occur, and the attacker generates a logical error in the result of the cryptographic operation. A variant consists in suddenly modifying the clock period.

There is another way to generate faults in cryptographic devices without physically modifying the circuit. The idea is to place an electromagnetic sensor above the device and to generate an electromagnetic field through the chip. Although few works have reported concrete results of this kind of attack, concrete experimental results can be found in [61].

Some circuits can even be disturbed by a steady stress. For instance, paper [34] shows how to obtain incorrect computations from an AES implemented in an FPGA.

The method consists in underfeeding the device, so as to generate setup time violations. A comprehensive study of this way of injecting such faults in an FPGA is available in [15] for various FPGA families. The outcome of these experiments is that the description of the algorithm can greatly improve the resistance of the implementations against these "global faults".

In the rest of this chapter, we only deal with Power and Electromagnetic based Attacks will be detailed [67]. For more information on Timing Attacks, the reader can consult [35] or [19]. For information on Fault Attacks, [16, 17, 30] and [12] are good starting points.

The following explanations are applicable to for all kind of ICs used in embedded devices, and especially for FPGAs that use hardware or software cryptographic primitives.

3.2 Power and Electromagnetic Measurement Platform

In this section, we describe the platform used to perform power and/or electromagnetic measurements of an FPGA running cryptographic operations.

3.2.1 Equipment

One example of an overall platform is given in Fig. 3.5, it is made up of several parts described in the following sections. Other boards are described in [67]. Also of particular interest are the so-called "Side Channel Analysis Standard Evaluation BOards" (SASEBO) [54].

3.2.1.1 Attacked Device

For the experiments described in this chapter, the attacked device is an Xilinx FPGA Spartan3-1000 embedded in a Digilent board. The Spartan die is encapsulated in a Cavity Up Ball Grid Array (BGA) package, and is roughly 7 mm by 7 mm in size, whereas the whole package is roughly 17 mm by 17 mm.

3.2.1.2 Oscilloscope

The main component of the equipment required for a Side Channel Attack, (and the most expensive device in a Side Channel Attack platform) is an oscilloscope. A Lecroy 735Zi scope was used for the experiments described in this chapter. It belongs to the latest generation of Lecroy scopes, and has impressive properties:

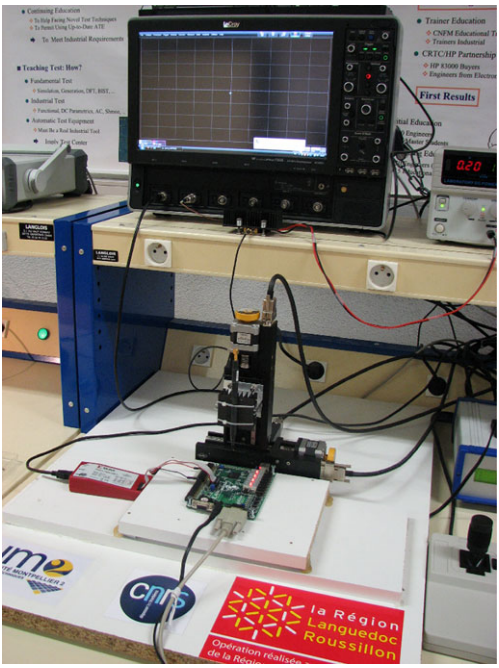


Fig. 3.5 Power and Electromagnetic Measurement Platform

- 4 channels;
- a maximum sampling rate of 40 GSa/s;
- a frequency bandwidth of 3.5 GHz;
- a resolution of 8 bits per point (without averaging);
- a vertical sensitivity of 2 mV;
- a maximum memory depth of 32 MSa.

Moreover, like other high-performance scopes, it allows remote control via Ethernet connections. Thus, once the scope is on, it can be fully controlled remotely. Furthermore, specific advanced modes for the trigger can be very useful for SCA purposes, but these functionalities were not exploited in our experiments.

3.2.1.3 Current Probes

To accurately measure the current consumed by the attacked device, current probes have interesting properties and are affordable. Tektronix current probes CT1 and CT6 [1] were used in the platform described here. First, the voltage regulator on the Digilent board supplying the power to the FPGA core was removed. Power was supplied by a battery, and the current probe was placed on the VDD or GND wire (between the battery and the pins of the core).

The current probe CT1 has a frequency bandwidth of 1 GHz and the CT6 of 2 GHz, and an accuracy of 5 mV/mA. Low-noise amplifier (described below) was

plugged at the output of the current probe to increase the amplitude of the signal enabling accurate measurement of power consumption.

3.2.1.4 Magnetic Sensors

The size and the shape of a magnetic sensor are the most important characteristics that determine the accuracy of the sensor. Hand-made magnetic sensors can be built. Attempts have often been made to build the smallest coiled wire, made of copper, and with several turns. Although these sensors produce good results (for instance some DEMA attacks succeeded with several hundred measurements), commercial magnetic sensors are often preferred [55].

Consequently, we used the Rhode and Schwarz HZ-15 probe set which contains different electric and magnetic sensors. Among them, the H field probe $RSH2.5\text{-}2$ gives accurate measurements, with good precision (its diameter is around 0.5 mm, thus when placed close to the IC, one is about 1 mm when working on encapsulated chips). Moreover, it allows precise measurements with a huge frequency bandwidth (spanning from 1 MHz to 3 GHz) even through the package of the FPGA. It consequently appears to be a good compromise to measure the magnetic activity of packaged chips.

3.2.1.5 Low-Noise Amplifier

To ensure sufficient power and accurate electromagnetic measurements, a low noise amplifier can be used. We used one with a frequency bandwidth of 1 GHz, and a gain of 63 dB.

3.2.1.6 Motorized X-Y-Z Stage

Using a tiny magnetic sensor to measure a particular area of the attacked chip requires positioning the sensor at the exact location required. For this task, we used a motorized X-Y-Z stage. The three axes have a precision of 50 µm, and the stage can be fully controlled remotely by serial connection.

3.2.1.7 Computer

A personal computer was used to control the whole platform. It was connected to the board embedding the attacked device via one serial port, to the scope via Ethernet, and to the X-Y-Z motorized stage via another serial port to drive it.

3.2.2 Acquisition Software

To automate the acquisition of measurements, we developed a software in Matlab. It runs on the computer controlling the whole platform. Typically, to perform an EM cartography, i.e. to obtain EM measurements of the same calculation(s) with the magnetic sensor placed at different positions above the chip to draw a map, the following steps are recommended:

- the initial position (X_{init}, Y_{init}, Z_{init}), the number of positions (nb_X, nb_Y), the displacement step (dis), the key (K) and the number of plaintexts (nb_{PTIs}) are chosen by the user;
- once the script is launched, *PTIs* are chosen randomly;
- the motorized stage is positioned at the first position (X_{init}, Y_{init}, Z_{init});
- the computer sends to the scope and adjusts it so that the entire computation is correctly recorded; it also sends the key K (optional) and the PTI_1 to the chip; the chip sends to the scope a trigger signal on the channel 1 to launch the recording of the magnetic signal on channel 2 (the magnetic sensor is connected to the low noise amplifier, which is connected to channel 2 of the scope);
- once the computation is finished, the scope sends the acquired EM trace to computer;
- the two previous steps are repeated nb_{PTIs} times;
- the motorized stage is positioned at the second position ($X_{init} + dis$, Y_{init}, Z_{init}), and the three previous steps are repeated;
- the same steps are repeated until all the positions have been covered.

The localization of the cryptographic modules is an advanced topic that is beyond the scope of this introduction concerned with side channel attacks. Interested readers should refer to [23, 52, 56, 57].

3.3 Leakage Models

Power and Electromagnetic (EM) based Side Channel Attacks is a hardware cryptanalytic technique that exploits physical leakages emitted by a cryptographic device. In this section, we explain how physical measurements can be realized and how leakage model can be used.

3.3.1 Power Consumption Leakage

Complementary Metal-Oxide-Semiconductor (CMOS) is the most used technology in microprocessors, microcontrollers, static RAM, and other digital logic circuits. Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn while the transistors in

the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, like NMOS. CMOS also enables a high density of logic functions on a chip. It was primarily for these reasons that CMOS won the race in the 1980s and became the most widely used technology for VLSI chips.

The power consumption of a CMOS circuit is the sum of the power consumption of the logic cells that make up the circuit [31]. Hence, the total power consumption mainly depends on the number of logic cells in a circuit, the connections between them, and how the cells are built. The power consumption of a CMOS gate is generally considered in terms of two components [22]:

- *The dynamic power* component: this is mainly related to the charging and discharging of the load capacitance at the gate output, but also to the short circuit current that flows during the transition of the input from one voltage level to another. Indeed there is a short period during which both PMOS and NMOS transistors are on simultaneously, thus creating an electrical path between the VSS and VDD rails.
- *The static power* component: this is due to leakage in the substrate that flows even when the gate is not switching. In turn, this leakage is made up of several components including gate to source leakage, which flows directly through the gate insulator, mostly by tunneling, and source-drain leakage attributed to both tunneling and sub-threshold conduction. However, so far no research paper has reported an attack using this static leakage.

While the transistors that comprise the CMOS gate are switching between on and off states, a significant amount of power is consumed, due to the dynamic power consumption. This characteristic of the CMOS technology is interesting from a cryptanalytic point of view. Indeed, an attacker can use the fact that the dynamic power consumption is tightly correlated with the number of switching bits to guess a secret value in a cryptographic operation.

Moreover, electrical simulations in recent works [39] showed that, in deep-submicron CMOS technologies, the percentage of static power consumption among the global power consumption of a circuit increases with a decrease in the technology shrinking techniques used. As the static power consumption of a gate mainly depends on the biasing of its inputs and its internal nodes, static power consumption is correlated with the last value computed by this gate. An attacker can thus exploit the relation between static power consumption and a computed value to guess a secret value in a cryptographic computation [51].

Different practical techniques can be used to measure the power consumption of a chip, depending mainly on how the chip is encapsulated, and how it is supplied:

- if a small resistor is placed between the VDD (GND) of the circuit and the VDD (GND) of the power supply, the consumption of the circuit can be measured at the pins of the resistor;
- another way is to use a dedicated current probe [1].

3.3.2 Electromagnetic Leakage

EM radiations emitted by an integrated circuit are mainly due to the displacement of current through the rails of the metal layers. This phenomenon has been formalized by Maxwell's equations.

Practical experiments in [47] showed that the power and ground (P/G) network represent the majority of EM radiations of a CMOS device. Thus, when transistors are switching between on and off states, current crosses the P/G network to supply the pins of the switching transistors, and this displacement of current creates variations in the EM field induced by the electrical behavior of the circuit.

Practically speaking, either the electric or the magnetic field can be measured. But for EM based SCA, measuring the magnetic field generally gives better results. The magnetic field emitted by a chip is measured using a near field magnetic sensor [48].

3.3.3 Hamming Weight vs. Hamming Distance Models

In Kocher's original paper [36], the leakage model used is based on the Hamming Weight of the bit that the attacker tries to guess. This model, called the *Hamming Weight leakage model*, considers that a 0 does not lead to excess of power consumption (or EM radiations), whereas a 1 involves a significant amount of power consumption. Thus:

- transitions $0 \rightarrow 0$ and $1 \rightarrow 0$ are considered as not leading to excess power consumption;
- transitions $0 \rightarrow 1$ and $1 \rightarrow 1$ are assumed involve an excess of power consumption.

Actually this model does not exactly match with the reality, except in precharged logics (where each register is precharged at 0 before being updated, in this case the Hamming Distance of a such word is equal to its Hamming Weight).

So the *Hamming Weight (HW) leakage model* could be improved by considering the switching state rather than the output state of the word concerned [18] (the Hamming Distance between the previous and the new state):

- transitions $0 \rightarrow 0$ and $1 \rightarrow 1$ do not lead to excess power consumption;
- transitions $0 \rightarrow 1$ and $1 \rightarrow 0$ involve excess power consumption.

This leakage model is called the *Hamming Distance (HD) leakage model*. Figure 3.6 summarizes how these two leakage models rank transitions of one bit, considering its four possible transitions.

Other leakage models are possible, for instance a leakage model that distinguishes rising transitions and falling transitions has been studied in [45, 48], but the results obtained were similar to those obtained with the HD model.

	HD = 0	HD = 1
HW = 0	0 ➔ 0	1 ➔ 0
HW = 1	1 ➔ 1	0 ➔ 1

Fig. 3.6 Ranking of a transition of one bit following the Hamming Weight and Hamming Distance leakage models

3.4 Side-Channel Attacks

As shown previously, both the power consumption and the EM radiations of a circuit are tightly correlated with the data it processes. In this context, different cryptanalytic methods have been proposed to exploit this behavior, and to guess secrets involved in a cryptographic operation. The four main methods that exploit power or EM leakage presented here are respectively SPA/SEMA, DPA/DEMA, Template attacks, and Information theoretic attacks.

Note that, in the rest of the chapter, we do not distinguish between power consumption and EM radiations of cryptographic operations. Our explanations focus on the algorithmic treatment of physical leakages, mainly considering power based attacks. Indeed, up to now most published works are related to the power side channel but the algorithmic treatment is the same as for the EM side channel.

3.4.1 SPA/SEMA

The Simple Power Analysis (SPA), originally described in [36], exploits a single power consumption trace of a cryptographic operation to guess the secret used in the computation (SEMA stands for Simple ElectroMagnetic Analysis, in the case of an EM trace of a cryptographic operation).

Actually, this method works directly when applied on some asymmetric cryptography operations. For instance, in the case of an RSA decryption, the computation of the modular exponentiation:

$$m = c^d \pmod{n} \tag{3.1}$$

with m being the plaintext, c the ciphertext, d the private key exponent and n the modulus, can be naively calculated using the right-to-left binary method (d_i is the ith bit of d), as described in Algorithm 1.

Depending the value of each bit of the private key exponent d, one computes only a squaring operation or both a multiplication and a squaring operation. Hence the power consumption (or EM radiations) of each elements of the *for* loop will be different depending on the value of the bit of the private key exponent d, and by simply analyzing one trace, an attacker can easily guess bit values of the private key d. Figure 3.7 shows a typical trace (after demodulation at the FPGA clock frequency [44]).

A similar attack is possible on ECC, but rather than attacking the square and multiply algorithm, one can attack the double-and-add algorithm involved in the point multiplication.

Algorithm 1 Right-to-left binary method for modular exponentiation

Inputs: c, $d = (d_{l-1}d_{l-2}\ldots d_0)_2$, n
Output: m
$m = 1$
for $i = 0$ to $l - 1$ **do**
  if $d_i = 1$ **then**
    $m = m * c \pmod n$
  end if
  $c = (c * c) \pmod n$
end for
Result: m

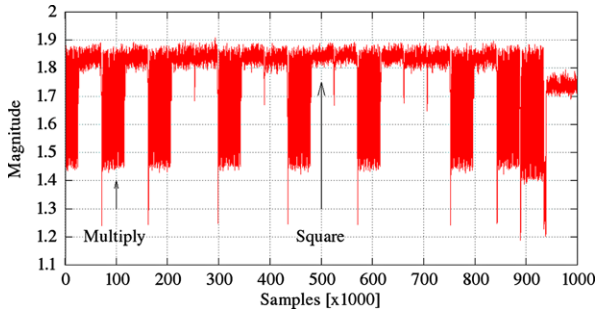


Fig. 3.7 Example of an SEMA on RSA

Concerning symmetric key cryptography, SPA attacks focusing on specific implementations of the key schedule have been published. [42] reports an SPA against the AES Key Expansion, and [5] describes an SPA against the key schedule of the Camellia block cipher.

3.4.2 DPA/DEMA

The Differential Power Analysis (DPA), originally described in the Kocher's pioneering paper [36], is a powerful technique that allows the attacker to guess secret keys used in a lot of cryptographic primitives. A lot of authors have proposed improvements of this attack along with countermeasures. When the attacker uses the EM radiations of the chip rather than its power consumption, DPA becomes DEMA, for Differential Electromagnetic Analysis.

3.4.2.1 Original Algorithm

DPA is a known plaintext or known ciphertext attack. The adversary enciphers (resp. deciphers) N PlainText Inputs (*PTI*) (resp. N CipherText Outputs, *CTO*) with an unknown key stored in the device, and monitors the power consumption of the device

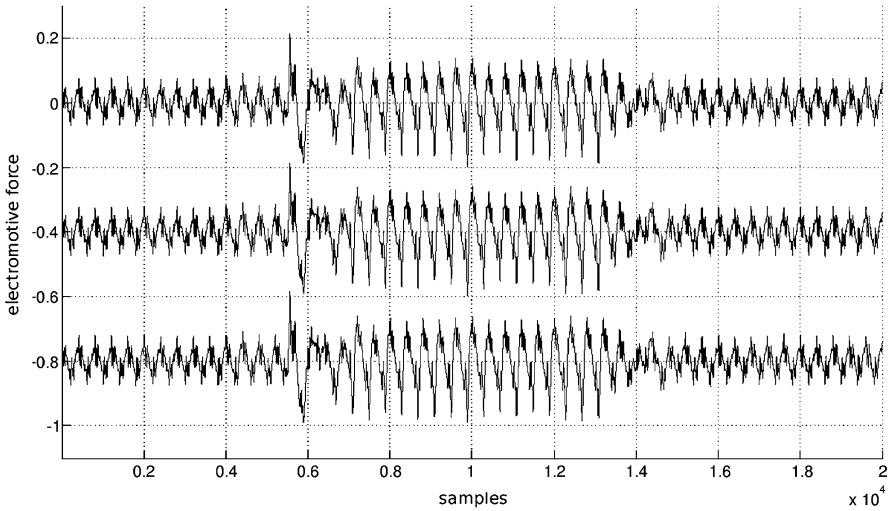


Fig. 3.8 Electromagnetic radiation traces of three different DES encryptions monitored on an FPGA

during each ciphering (resp. deciphering) operation. At the end of the first stage, he obtains *N PTIs* (resp. *N CTOs*) and *N* power consumption traces. Each trace is the change versus time in the power consumption of the chip.

Note that these traces have to be well aligned; this means that the time index of the beginning of ciphering has to be the same for all measurements.

If the measurements are not well aligned (due for instance to countermeasures like random clock frequency or dummy instructions), different preprocessing techniques enables traces to be re-synchronized [4, 6, 7, 21, 33, 53].

Figure 3.8 shows several EM radiation traces corresponding to the DES encryptions of different *PTIs* with the same key monitored on an FPGA.

The second stage is statistical processing of the *N PTIs* (resp. *N CTOs*) with the *N* traces.

In the rest of the chapter, the DES [2] is used as example, because it is the best most known block cipher and the principles that apply to the DPA also apply to other cryptographic ciphers, such as AES [3]. For the sake of convenience, we consider that the adversary is using a known plaintext attack and is trying to guess the round-key 1 of the DES (the remaining 8 bits could then be discovered through a brute force attack). A similar algorithm allows the round-key 16 in a known ciphertext attack to be discovered.

Since the set of all possible values for the round-key 1 is too large to test all of them, the adversary usually divides the round-key 1 into 8 parts of 6 bits each (here called sub-keys) and attacks each sub-key independently and sequentially. Thus, for each sub-key, there are 64 possible values.

Moreover, attacking each sub-key independently allows all kind of implementations to be targeted. For instance, in software implementations, depending on the size of the data bus (generally 8, 16 or 32), sbox processing can be computed se-

quentially, and so sbox 1 is not processed at the same time index as sbox 8. Unlike in some hardware implementations, all the sboxes are processed at the same time, meaning other approaches are possible, as explained below.

The adversary makes hypothesizes on the 6 bits of the attacked sub-key, and for each *PTI*, he computes the output (4 bits) of the corresponding sbox. This value is called the Intermediate Value (*IV*). In the state-of-the-art mono-bit DPA [36], the adversary targets out of the four, for instance the Less Significant Bit (LSB).

If the LSB of IV_1 (corresponding to the first plaintext, PTI_1) is equal to 0, the associated trace (T_1) is ranked in set A. If the LSB of IV_1 is equal to 1, T_1 is ranked in set B. As explained above, the adversary ranks all the traces, in sets A or B, and then computes the difference in the means of sets A and B. The resulting curve is called a differential curve, and corresponds to a sub-key hypothesis. The adversary computes the 64 differential curves corresponding to the 64 possible values for a given sub-key hypothesis.

As explained in [36], the differential curve, denoted Δ, for a sub-key hypothesis K_s, is calculated as follows:

$$\Delta_{K_s}[j] = \frac{\sum_{i=1}^{N} D(PTI_i, K_s)T_i[j]}{\sum_{i=1}^{N} D(PTI_i, K_s)} - \frac{\sum_{i=1}^{N}(1 - D(PTI_i, K_s))T_i[j]}{\sum_{i=1}^{N}(1 - D(PTI_i, K_s))}, \tag{3.2}$$

where $\Delta_{K_s}[j]$ is the jth sample of the differential curve, N is the number of traces used, PTI_i is the ith plaintext, $T_i[j]$ is the jth sample of the trace and D the decision function than ranks traces in set A or B, also called *selection function*.

If the sub-key hypothesis is wrong, all the computed intermediate values will be wrong with respect to the data really processed on the chip. In this case, the traces will be randomly classified in sets A and B. The mean curves of sets A and B will be similar, and the differential curve will look like a thick horizontal line (mainly composed of noise).

On the other hand, if the sub-key hypothesis is correct, all the computed intermediate values will match the data really processed on the chip, and traces in set A will have the same characteristic: at the time index where the intermediate value is computed, the LSB of *IV* equal to 0 will not lead to excess power consumption. Conversely, when the LSB of *IV* is equal to 1, a bit more energy will be consumed at the same time index and spikes corresponding to the clock cycle where *IV* is computed will be greater on traces in set B than on traces in set A. When the difference in the means of the 2 sets is being computed, a spike will appear at the time index of the differential curve concerned indicating that the sub-key hypothesis is correct.

Figure 3.9 shows the 64 differential curves computed following guesses of sub-key 1 of round-key 1 of the DES, using 500 EM traces (each one averaged 20 times). Differential curves corresponding to wrong guesses of the sub-key are in cyan, whereas the curve corresponding to the correct guess of the sub-key is in black, and has the greatest peak. To guess the eight parts of round-key 1, processing is applied sequentially on each sub-key.

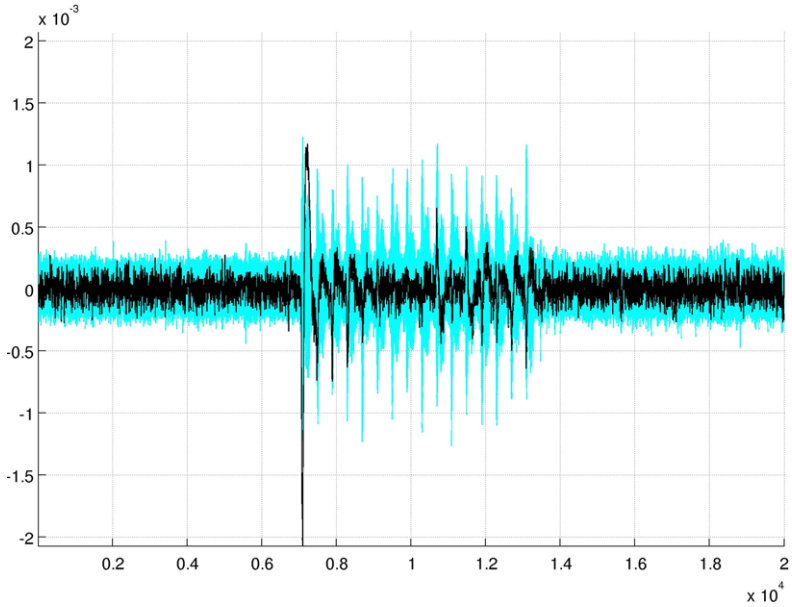


Fig. 3.9 Example of a successful DEMA: 64 differential curves computed following hypotheses of sub-key 1 of round-key 1 of the DES, using 500 EM traces (each one averaged 20 times). Differential curves corresponding to wrong hypotheses of the sub-key are in *cyan*, the curve corresponding to the correct hypothesis of the sub-key is in *black* (color online)

3.4.2.2 Improvements of DPA

Hamming Weight vs. Hamming Distance Leakage Models In the original DPA algorithm, and as explained in the previous section, the selection function follows the Hamming Weight (HW) power consumption model. But previously, we explained that the Hamming Distance (HD) power consumption model matches reality better.

Moreover, an iterative implementation is used in most hardware implementations of block ciphers. Thus, only one generic round is implemented, and at each clock cycle the intermediate result of the previous round is used as input with the current round-key generated by the key schedule.

Hence, rather than considering an output bit of an sbox, we can consider the bit linked to the previous one in R1 [2]. Thus, as the adversary knows the value of this bit in R0 (because he/she knows the PTI), and assuming that R0 and R1 are stored in the same register which is updated at each round, we can compute the Hamming Distance between the value of the targeted bit in R0 and R1. In this case, it is generally not the power consumption of an inverter that is estimated, but the power consumption of a flip-flop.

Multi-bit DPA Another way to improve DPA attacks consists in considering the four bits linked to the output of the sbox rather than only one. As proposed in [14],

we can compute the differential curve for each bit out of the four from the output of the sbox, and sum the four differential curves.

We can also rank traces using another criterion [43]: by summing the number of switching bits out of the four considered, and if the sum is smaller than 2, we rank the associated trace in set A, whereas if the sum is greater than 2, we rank the trace in set B.

Distinguishers In the original DPA algorithm, the *distinguisher* used to correlate predictions with measurements is called *Difference of Mean* (DoM). Different works have proposed other methods, based on different mathematical tools. Here, we describe the best known.

Partition Distinguishers As explained previously, the *Difference of Means* (DoM) distinguisher consists in ranking, according to a key hypothesis, all the traces in two sets, following a criterion. Then the difference in the means of the two sets of traces is computed. Applying this method to the different key hypotheses reveals the correct key hypothesis.

In the previous paragraph, we described different improvements of Kocher's attack, like the multi-bit DPA proposed by Messerges [43], or the method suggested by Bevan [14].

Another proposal, presented in [37], consists in ranking traces in more than two sets. In the case of attacking a DES, if the attacker focuses on the four output bits of an sbox, he/she can rank traces in five sets, one per Hamming Weight. Considering four bits b_0, b_1, b_2 and b_3, the Hamming Weight of the word $b_3b_2b_1b_0$ is between 0 and 4, and leads to five possible values. Then the attacker has to choose coefficients a_i, $i = 0, \ldots, 3$, assigned to each set. This method has been called Partitioning Power Analysis (PPA).

Unfortunately, no efficient method has been proposed to determine these coefficients in the general context. Indeed, due to the process variations, foundries cannot ensure that all the bus lines have *exactly* the same geometrical dimensions. For example, bit b_0 will not consume exactly the same amount of power (or radiate exactly the same field) as bit b_1. This is all the more true in the presence of countermeasures. For instance, in a circuit where the nets are balanced (for instance thanks to a dual-rail logic [68]), the coefficients a_i are given in Fig. 3.10 [58].

Blind techniques enable identification of the coefficients. The optimal coefficients in the context of masking are derived in [50]. In a general context, the stochastic approach, discussed in Sect. 3.4.3.2, always applies.

Comparison Distinguishers In a differential SCA, the attacker wants to estimate the relation between predictions, depending on a key hypothesis, and on measurements. In [18], the authors suggest using a well-known statistical tool, *Pearson's correlation coefficient*. Let X and Y be two random variables, $\mathrm{cov}(X, Y)$ be the covariance between X and Y, and σ_X and σ_Y be the standard deviations of X and Y, *Pearson's correlation coefficient* is computed as follows:

$$\rho(X, Y) = \frac{\mathrm{cov}(X, Y)}{\sigma_X \times \sigma_Y}. \tag{3.3}$$

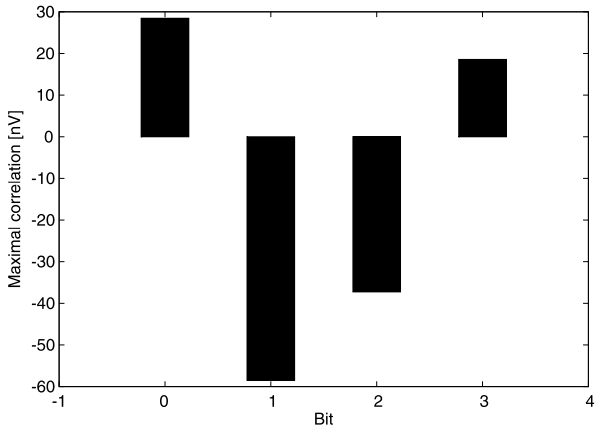


Fig. 3.10 Optimal weights found for the sbox output of DES in dual rail with precharge logic

This mathematical tool measures the dependence between two quantities, and for SCA, its application is straightforward. Let N be the number of traces, K_s the key hypothesis, PTI_i the ith PlainText Input, $H(K_s, PTI_i)$ the number of switching bits according to the key hypothesis K_s and the plaintext PTI_i, $T_i(j)$ the jth sample of the ith trace, Eq. (3.3) becomes:

$$\rho_{K_s}[j] = \frac{N.\sum_{i=1}^{N} H(K_s, PTI_i).T_i(j) -}{\sqrt{N.\sum_{i=1}^{N} H(K_s, PTI_i)^2 - (\sum_{i=1}^{N} H(K_s, PTI_i))^2}} \times \frac{\sum_{i=1}^{N} H(K_s, PTI_i).\sum_{i=1}^{N} T_i(j)}{\sqrt{N.\sum_{i=1}^{N} T_i(j)^2 - (\sum_{i=1}^{N} T_i(j))^2}}. \tag{3.4}$$

The variant of the DPA using the *Pearson's correlation* has been called Correlation Power Analysis (CPA), and was first introduced in [18].

However, *Pearson's correlation* measures only linear dependencies between two quantities. Other tools, like Spearman's rank correlation, or Kendall's tau rank correlation, can measure linear dependencies between two random variables, but practical experiments have shown that these tools give very similar results [9].

Another author [28] introduced a different statistical tool, mutual information. Mutual information allows both linear and non-linear dependencies between two random variables to be measured. This is called Mutual Information Analysis (MIA). This technique is discussed in Sect. 3.4.4.1.

Key Search Generally, when the attacker has computed the 64 differential curves corresponding to guesses of the sub-key S_k using N traces, he/she searches for the maximum (in absolute) of each differential curve. Among the 64 maxima, the biggest one reveals the guessed during the attack using N traces. A classical optimization for the key search step consists in reducing the proportion of the differential curve inspected to detect the maximum. The key search may concentrate on a part of the curve corresponding to the attacked round.

3.4.3 Template Attack

The Template Attack (TA) is considered to be the most powerful type of SCA. The scenario of such an attack is slightly different than that required for other kinds of SCA. First, to build *templates* for the different possible values of the secret, attackers have to be in possession of a circuit identical to the circuit under attack. Then, they measure one trace on the attacked device, and using appropriate methods, compare this trace with the different templates they built to guess the value of the secret.

3.4.3.1 Original Template Attack

This attack, firstly described in [20], happens in two stages, the template building stage, and the template matching stage.

Template Building Phase In the first stage, attackers use the circuit under their full control, which is identical to the circuit under attack, to build templates for each pair of data and key.

Power traces can be characterized by a multivariate normal distribution, which is fully defined by a mean vector M and a covariance matrix C. In the rest of the paragraph, the pair (M, C) is referred to as a *template*. The attacker builds a *template* $(M, C)^{d_i,k_j}$ for each pair of data and key (d_i, k_j) following these steps:

- for each pair of data and key (d_i, k_j), acquire p traces $T_1^{d_i,k_j}, \ldots, T_p^{d_i,k_j}$;
- compute a mean vector M^{d_i,k_j} for each pair (d_i, k_j) from the p traces as follows:

$$M^{d_i,k_j} = \frac{1}{p}\sum_{l=1}^{p} T_l^{d_i,k_j}; \tag{3.5}$$

- (optional) compute pairwise differences between the mean vectors M^{d_i,k_j} in order to identify and select only points $P_1, \ldots, P_n$ at which large differences show up. This optional step significantly reduces the processing overhead with only a small loss of accuracy;
- for each pair d_i, k_j and for each acquired trace $T_l^{d_i,k_j}$, $l = 1, \ldots, p$, compute the noise vector $N_l^{d_i,k_j}$, $l = 1, \ldots, p$ as follows:

$$N_l^{d_i,k_j} = (T_l^{d_i,k_j}(P_1) - M^{d_i,k_j}(P_1) \ldots T_l^{d_i,k_j}(P_n) - M^{d_i,k_j}(P_n)); \tag{3.6}$$

- for each pair d_i, k_j, compute the noise covariance matrix C^{d_i,k_j} between all pairs of components of the noise vectors $N_l^{d_i,k_j}s$, $l = 1, \ldots, p$, for all the p traces, as follows:

$$C^{d_i,k_j}[u, v] = \mathrm{cov}(N_l(P_u), N_l(P_v)). \tag{3.7}$$

Thus, for each data and key pair (d_i, k_j), the attacker has built a *template*, that characterizes the deterministic part of the power trace, but also its noise part, modeling it using a multivariate Gaussian model.

Template Matching Phase In the second stage, the attackers use a power trace from the device under attack to determine the key. To this end, they evaluate the probability density function of the multivariate normal distribution with $(M, C)^{d_i,k_j}$ and the trace of the device under attack.

In other words, given a power trace t of the chip under attack, and a *template* $(M, C)^{d_i,k_j}$, the attacker computes the probability:

$$p(t; (M, C)^{d_i,k_j}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(C)}} e^{-\frac{1}{2}(t-M)^T \cdot C^{-1} \cdot (t-M)}. \tag{3.8}$$

The attacker does this for every template. As a result, he/she obtains the probabilities $p(t; (M, C)^{d_1,k_1}), \ldots, p(t; (M, C)^{d_D,k_K})$. The probabilities measure how well the templates fit to a given trace. Intuitively, the highest probability should indicate the correct template. Because each template is associated with a key, the attacker also gets a clue to the correct key. This intuition is also supported by the maximum-likelihood decision rule.

The points of interest can be either selected heuristically [20], by *ad hoc* techniques [27], or by dimensionality reduction tools, such as principal components analysis [11].

3.4.3.2 Stochastic Method

An improvement of TA is suggested in [59, 60]. It consists in using a stochastic method to approximate the real leakage function within a suitable vector subspace. The attack requires some engineering skills to introduce a relevant parametric model. Then, the on-line attack basically consists in simultaneously:

- estimating the best parameters, using a linear regression, and
- deciding which model is the closest to the side channel measurements, using the minimal Euclidean distance as a distinguisher.

An empirical comparison of template and stochastic attacks is provided in [65].

3.4.4 Information Theoretic Attacks

3.4.4.1 MIA—Mutual Information Analysis

At CHES 2008, a generic side channel distinguisher, MIA, was proposed in [29]. It is an attractive alternative to the above-mentioned attacks since some assumptions about the adversary can be disregarded. In particular, it does not require a linear dependency between the leakage and the predicted data, as is the case for DPA and CPA, and so it is not only able to exploit any kind of dependency but does not need to profile the leakage as it is the case for template attacks [20].

The rationale of MIA is to compare distributions of observations with random distributions rather observations with a model. This is why the MIA measures the Kullback-Leibler divergence between observations knowing the correct sub-key and observations not knowing anything about the internal values [70].

3.4.4.2 EPA—Entropy Power Analysis

Mutual information analysis has been tested in noisy real world designs. It indeed appears to be a powerful approach to break unprotected implementations. However, the MIA fails when applied on a DES cryptoprocessor with masked substitution boxes (sboxes) in ROM [66]. However, this masking implementation remains sensitive to Higher Order Differential Power Analysis (HO-DPA). For instance, an attack based on variance analysis clearly reveals the vulnerabilities of a first-order masking countermeasure. A novel approach to information theoretic HO attacks, called Entropy-based Power Analysis (EPA), has therefore been proposed. This new attack gives greater importance to highly informative partitions and distinguishes the key hypotheses better [41].

3.4.4.3 VPA—Variance Power Analysis

Information theoretic attacks are extremely powerful, as they exploit any deviation from a random probability density function (PDF). However, estimating PDFs is a hard task [49, 70], since for the PDFs to be accurate, it ideally requires a lot of measurements. To simplify the attack, articles [38, 40, 64] suggest limiting the analysis of the PDFs to their second cumulant. The estimation is much faster and the distinguisher thus gains strength. This is referred to as variance power analysis (VPA). This metric to compute distances between PDFs is extremely useful against implementations protected by first-order masking.

3.5 Conclusions

More than ten years after the first publication of Kocher's attack, a lot of improvements have been proposed for power and EM based Side Channel Attacks. Firstly, it has been shown that both power consumption and EM radiations of a circuit can leak sensitive information. Secondly, different methods have been proposed to extract a secret key used in a cryptographic operation from physical leakages emitted by the device. In this chapter we described SPA/SEMA, DPA/DEMA and Template Attacks. We also described different improvements of differential SCA that give better results than the original attack, especially when the attacker has some knowledge of the implementation. To thwart these attacks, a lot of methods have been proposed. Due to the specificities of FPGAs, some of these countermeasures are not applicable, while others are relatively easy to integrate. The next chapter describes the main methods used to protect cryptographic algorithms implemented in FPGAs against Side Channel Attacks.

References

1. Tektronix Current Probes Ct1, Ct2, Ct6. http://www.tek.com

2. Data Encryption Standard: FIPS PUB 46-3 (1999)
3. Advanced Encryption Standard: FIPS PUB 197 (2001)
4. A method for resynchronizing a random clock on smartcards. In: Eurosmart (2001)
5. A simple power analysis attack against the key schedule of the camellia block cipher. Inf. Process. Lett. **95**(3), 409–412 (2005)
6. Improving the DPA attack using wavelet transform. In: NIST Physical Security Testing Workshop (2005)
7. High-resolution side-channel attack using phase-based waveform matching. In: CHES, pp. 187–200 (2006)
8. Diode Laser Station. Riscure (2009)
9. DPA contest 2008/2009. http://www.dpacontest.org (2009)
10. Anderson, R., Kuhn, M.: Low cost attacks on tamper resistant devices. In: Proceedings of the 5th International Workshop on Security Protocols, pp. 125–136 (1998)
11. Archambeau, C., Peeters, É., Standaert, F.-X., Quisquater, J.-J.: Template attacks in principal subspaces. In: CHES, Yokohama, Japan, October 10–13. LNCS, vol. 4249, pp. 1–14. Springer, Berlin (2006)
12. Bar-el, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks (2004)
13. Di-Battista, J., Courrège, J.-C., Rouzeyre, B., Torres, L., Perdu, P.: When failure analysis meets side-channel attacks. In: CHES, pp. 188–202 (2010). doi:10.1007/978-3-642-15031-9_13
14. Bevan, R., Knudsen, E.: Ways to enhance differential power analysis. In: ICISC, pp. 327–342 (2002)
15. Bhasin, S., Selmane, N., Guilley, S., Danger, J.-L.: Security evaluation of different AES implementations against practical setup time violation attacks in FPGAs. In: HOST (Hardware Oriented Security and Trust), July 27th, pp. 15–21. IEEE Comput. Soc., Los Alamitos (2009). doi:10.1109/HST.2009.5225057. In conjunction with DAC-2009, Moscone Center, San Francisco, CA, USA
16. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: CRYPTO, pp. 513–525 (1997)
17. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: EUROCRYPT, pp. 37–51 (1997)
18. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: CHES, pp. 16–29 (2004)
19. Brumley, D., Boneh, D.: Remote timing attacks are practical. In: SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium, pp. 1–1. USENIX Association, Berkeley (2003)
20. Chari, S., Rao, J., Rohatgi, P.: Template attacks. In: CHES, pp. 13–28 (2002)
21. Clavier, C., Coron, J.-S., Dabbous, N.: Differential power analysis in the presence of hardware countermeasures. In: CHES, pp. 252–263 (2000)
22. Coron, J.-S., Naccache, D., Kocher, P.: Statistics and secret leakage. ACM Trans. Embed. Comput. Syst. **3**(3), 492–508 (2004)
23. Dehbaoui, A., Lomne, V., Maurine, P., Torres, L.: Magnitude squared incoherence EM analysis for integrated cryptographic module localisation. Electron. Lett. **45**(15), 778–780 (2009). doi:10.1049/el.2009.0342
24. Dinur, I., Shamir, A.: Generic analysis of small cryptographic leaks. In: FDTC, Santa Barbara, CA, USA, August 21, pp. 51–65. IEEE Comput. Soc., Los Alamitos (2010). doi:10.1109/FDTC.2010.11
25. Dinur, I., Shamir, A.: Side channel cube attacks on block ciphers. Cryptology ePrint Archive, Report 2009/127. http://eprint.iacr.org/ (March 2009)
26. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: CHES, pp. 251–261 (2001)
27. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: CHES, Yokohama, Japan, October 10–13. LNCS, vol. 4249, pp. 15–29. Springer, Berlin (2006)

28. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: CHES, pp. 426–442 (2008)
29. Batina, L., Gierlichs, B., Prouff, E., Rivain, M., Standaert, F., Veyrat-Charvillon, N.: Mutual information analysis: a comprehesive study. J. Cryptol. **24**(2), pp. 269–291 (2010)
30. Giraud, C., Thiebeauld, H.: A survey on fault attacks. In: Smart Card Research and Advanced Applications VI, IFIP 18th, World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), Toulouse, France, 22–27 August, pp. 159–176. Kluwer, Dordrecht (2004)
31. Guilley, S., Hoogvorst, P., Pacalet, R.: Differential power analysis model and some results. In: Proceedings of WCC/CARDIS, Toulouse, France, August, pp. 127–142. Kluwer, Dordrecht (2004). doi:10.1007/1-4020-8147-2_9
32. Handschuh, H., Paillier, P., Stern, J.: Probing attacks on tamper-resistant devices. In: CHES, pp. 303–315 (1999)
33. Kafi, M., Guilley, S., Marcello, S., Naccache, D.: Deconvolving protected signals. In: ARES/CISIS, Fukuoka, Kyūshū, Japan, March 16th–19th, pp. 687–694. IEEE Comput. Soc., Los Alamitos (2009). doi:10.1109/ARES.2009.197
34. Khelil, F., Hamdi, M., Guilley, S., Danger, J.-L., Selmane, N.: Fault analysis attack on an FPGA AES implementation. In: NTMS, Tangier, Morocco, November, pp. 1–5. IEEE (2008). doi:10.1109/NTMS.2008.ECP.45
35. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, pp. 104–113. Springer, London (1996)
36. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO, pp. 388–397 (1999)
37. Le, T.-H., Clédiere, J., Canovas, C., Robisson, B., Serviere, C., Lacoume, J.-L.: A proposition for correlation power analysis enhancement. In: CHES, pp. 174–186 (2006)
38. Li, Y., Sakiyama, K., Batina, L., Nakatsu, D., Ohta, K.: Power variance analysis breaks a masked ASIC implementation of AES. In: DATE, Dresden, Germany, March 8–12, pp. 1059–1064. IEEE (2010)
39. Lin, L., Burleson, W.: Analysis and mitigation of process variation impacts on power-attack tolerance. In: DAC, pp. 238–243 (2009)
40. Maghrebi, H., Danger, J.-L., Flament, F., Guilley, S.: Evaluation of countermeasures implementation based on Boolean masking to thwart first and second order side-channel attacks. In: SCS, Jerba, Tunisia, pp. 1–6. IEEE (2009). Complete version online: http://hal.archives-ouvertes.fr/hal-00425523/en/. doi:10.1109/ICSCS.2009.5412597
41. Maghrebi, H., Guilley, S., Danger, J.-L., Flament, F.: Entropy-based power attack. In: HOST, Anaheim Convention Center, Anaheim, CA, USA, June 13–14, pp. 1–6. IEEE Comput. Soc., Los Alamitos (2010). doi:10.1109/HST.2010.5513124
42. Mangard, S.: A simple power-analysis (SPA) attack on implementations of the AES key expansion. In: ICISC, pp. 343–358 (2002)
43. Messerges, T., Dabbish, E., Sloan, R.: Investigations of power analysis attacks on smartcards. In: WOST, pp. 17–17 (1999)
44. Meynard, O., Rçal, D., Guilley, S., Danger, J.-L., Homma, N.: Enhancement of simple electromagnetic attacks by pre-characterization in frequency domain and demodulation techniques. In: DATE, Grenoble, France, March 14–18. IEEE Comput. Soc., Los Alamitos (2011)
45. Natale, G.D., Flottes, M.-L., Rouzeyre, B.: An integrated validation environment for differential power analysis. In: DELTA, pp. 527–532 (2008)
46. Nohl, K., Evans, D., Starbug, S., Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: Proceedings of the 17th Conference on Security Symposium, pp. 185–193. USENIX Association, Berkeley (2008). http://portal.acm.org/citation.cfm?id=1496711.1496724
47. Ordas, T., Lisart, M., Sicard, E., Maurine, P., Torres, L.: Near-field mapping system to scan in time domain the magnetic emissions of integrated circuits. In: PATMOS, pp. 229–236 (2008)
48. Peeters, E., Standaert, F.-X., Quisquater, J.-J.: Power and electromagnetic analysis: improved model, consequences and comparisons. Integration VLSI J. **40**, 52–60 (2007). doi:10.1016/j.vlsi.2005.12.013

49. Prouff, E., Rivain, M.: Theoretical and practical aspects of mutual information based side channel analysis. In: ACNS, Paris-Rocquencourt, France, June 2–5. LNCS, vol. 5536, pp. 499–518. Springer, Berlin (2009)
50. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)
51. Quisquater, J.-J., Standaert, F.-X.: Physically secure cryptographic computations: from micro to nano electronic devices. In: DSN, Workshop on Dependable and Secure Nanocomputing (WDSN), June 28. IEEE Comput. Soc., Edinburgh (2007). Invited Talk, 2 pages
52. Réal, D., Valette, F., Drissi, M.: Enhancing correlation electromagnetic attack using planar near-field cartography. In: DATE, Nice, France, April 20–24, pp. 628–633. IEEE (2009)
53. Réal, D., Canovas, C., Clédiere, J., Drissi, M., Valette, F.: Defeating classical hardware countermeasures: a new processing for side channel analysis. In: DATE, pp. 1274–1279 (2008)
54. Satoh, A.: Side-channel Attack Standard Evaluation Board, SASEBO. Project of the AIST—RCIS (Research Center for Information Security). http://www.rcis.aist.go.jp/special/SASEBO/
55. Sauvage, L.: Cartographie électromagnétique pour la cryptanalyse physique. PhD thesis, TELECOM-ParisTech, Paris, France (September 2009)
56. Sauvage, L., Guilley, S., Mathieu, Y.: Electromagnetic radiations of FPGAs: high spatial resolution cartography and attack of a cryptographic module. ACM Trans. Reconfigurable Technol. Syst. **2**(1), 1–24 (2009). Full text in http://hal.archives-ouvertes.fr/hal-00319164/en/. doi:10.1145/1502781.1502785
57. Sauvage, L., Guilley, S., Flament, F., Danger, J.-L., Mathieu, Y.: Cross-correlation cartography. In: ReConFig, Cancún, Quintana Roo, México, December 13–15, pp. 268–273. IEEE Comput. Soc., Los Alamitos (2010). doi:10.1109/ReConFig.2010.75
58. Sauvage, L., Nassar, M., Guilley, S., Flament, F., Danger, J.-L., Mathieu, Y.: Exploiting dual-output programmable blocks to balance secure dual-rail logics. Int. J. Reconfigurable Comput. **2010**, 375245 (2010). 12 pages. doi:10.1155/2010/375245
59. Schindler, W.: Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. J. Math. Cryptol. **2**(3), 291–310 (2008). ISSN (Online) 1862-2984, ISSN (Print) 1862-2976. doi:10.1515/JMC.2008.013
60. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: CHES, pp. 30–46 (2005)
61. Schmidt, J.-M., Hutter, M.: Optical and EM fault-attacks on CRT-based RSA: concrete results. In: Austrochip (2007)
62. Schmidt, J.-M., Kim, C.H.: A probing attack on AES, pp. 256–265 (2009)
63. Skorobogatov, S., Anderson, R.: Optical fault induction attacks. In: CHES, pp. 2–12 (2002)
64. Standaert, F.-X., Gierlichs, B., Verbauwhede, I.: Partition vs. comparison side-channel distinguishers: an empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In: ICISC, Seoul, Korea, December 3–5. LNCS, vol. 5461, pp. 253–267. Springer, Berlin (2008)
65. Standaert, F.-X., Koeune, F., Schindler, W.: How to compare profiled side-channel attacks? In: ACNS, Paris-Rocquencourt, France, June 2–5. LNCS, vol. 5536, pp. 485–498. Springer, Berlin (2009)
66. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J.: FPGA implementations of the DES and triple-DES masked against power analysis attacks. In: Proceedings of FPL 2006, Madrid, Spain, August. IEEE (2006)
67. Standaert, F.-X., Batina, L., Mulder, E.D., Lemke, K., Mentens, N., Oswald, E., Peeters, E.: Report on DPA and EMA Attacks on FPGAs. July 31 ECRYPT IST-2002-507932, "European Network of Excellence in Cryptography". Deliverable D.VAM.5. http://www.ecrypt.eu.org/ecrypt1/documents/D.VAM.5-1.pdf
68. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: DATE'04, Paris, France, February, pp. 246–251. IEEE Comput. Soc., Los Alamitos (2004). doi:10.1109/DATE.2004.1268856

69. Torrance, R., James, D.: The state-of-the-art in IC reverse engineering. In: CHES, pp. 363–381 (2009)
70. Veyrat-Charvillon, N., Standaert, F.-X.: Mutual information analysis: how, when and why? In: CHES, Lausanne, Switzerland, September 6–9. LNCS, vol. 5747, pp. 429–443. Springer, Berlin (2009)

Chapter 4
Countermeasures Against Physical Attacks in FPGAs

J.-L. Danger, S. Guilley, L. Barthe, and P. Benoit

Abstract This chapter presents a set of countermeasures against physical attacks specifically dedicated to FPGA. Countermeasures as masking, hiding, are first discussed. Then we give a set of information and an overview on different logic style designed to be robust against SCA. The main objective herein is to compare these techniques and show that they can be suitable and implementable for FPGA components. A comparison of different logic style will conclude this chapter.

4.1 Introduction

Off the shelf FPGAs are often used for high-end applications that require embedded cryptography. State of the art commercial FPGA technologies do not have any specific feature to withstand side channel attacks that target the user application. For this reason, methods to protect them have to be designed at the logic and back-end levels of the design stages. Many countermeasures have been developed for ASICs and this can be used as a basis to develop specific protections for FPGAs. However the regular FPGA tiling structure and the huge space of interconnect and programmable switches may limit or reduce the robustness of countermeasures originally designed for ASICs. In this chapter we provide some answers to attacks and countermeasures embeddable in off the shelf FPGAs, with a special focus on side-channel attacks in symmetrical cryptography.

The protection of cryptographic IPs against side channel attacks at logical level is currently not so advanced in FPGAS as it is in ASICs, even though at first sight, FPGAs appear to have an intrinsic structure which makes them much more vulnerable:

- ASIC protections at back-end level (such as the fat wires [59] or back-end duplication [20] methods) are hardly feasible in FPGAs because of constrained and *a priori* unknown or limited routing resources.
- The interconnection makes up the largest part of the FPGA [26]. It includes lines, pass transistors, transmission gates, bidirectional buffers, switch matrices

J.-L. Danger (✉)
Telecom Paris-Tech, Paris, France
e-mail: danger@enst.fr

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_4, © Springer Science+Business Media B.V. 2011

and connection boxes. These components increase the capacitive load of the interconnection and thus overall power consumption. This particularity facilitates passive attacks.

- D Flip-Flop (DFF) are numerous and fast. There are two reasons for their rapidity: they speed up processing and fight metastability. However they greatly increase power consumption increase. Moreover attacks on register nodes can lead to knowledge of the activity of combinatorial nodes that contain the secret information.
- The use of switches like pass transistors makes the power consumption model quite specific to FPGAs in which the current can vary according to a higher order equation in V_{dd} [13, 14].

With regards to ASICs, a study by Kuon and Rose [26] shows that FPGAs are on average 35 times bigger and consume 12 times more than ASICs. To withstand attacks on programmable devices that are not specifically designed for security, it could be worthwhile designing dedicated logic styles and implementation methods. In this chapter we provide an overview of current techniques and focus on different countermeasures (masking, hiding), and logic styles (WDDL, STTL, BCDL) specifically designed to fight Side Channel Attacks (SCA). Our aim is to show the potentials and limitations of countermeasures in FPGAs that could obstruct or prevent attacks.

4.2 Countermeasures Against Side Channel Attacks in FPGAs

The types of countermeasures that are used to protect hardware devices against SCAs can be considered at levels Protocol, Architecture and Netlist.

Protection of protocol and of architecture is independent of the technology (either software or hardware). However the implementation in FPGAs can take advantage of hardware properties like concurrent computation, reprogrammability and high levels of algorithmic noise.

The Protocol level can merely consist in regular key changes as described in [25, 32] in order to prevent the adversary from accumulating enough traces to be able to attack. One advantage of this level of protection is that it is provable. As an illustration of this technique, Fig. 4.1 shows that at every 100th ciphering operations the key is changed, thus limiting the scope of the attack if more than 100 traces are needed.

However this type of protection can be time consuming and costly as it requires a specific key exchange or synchronization mechanism to ensure Alice and Bob always use the same key. Rather than changing the key, the FPGA offers the possibility to reprogram the implementation. As mentioned by F.-X. Standaert [51], this feature certainly merits further research as only a few studies have been conducted so far. Among them Chaudhuri et al. [9] details a dedicated FPGA architecture using agility to thwart SCAs, whereas Mentens [33] explains how security can be enhanced with dynamic reconfiguration.

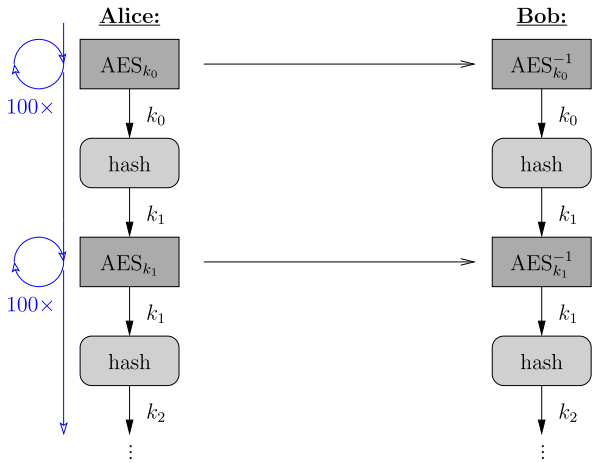


Fig. 4.1 Dynamic key change by hashing every 100th encryptions

Strengthening security at architectural level is certainly the least constraining method as it can be applied in all technologies that have logical model. For hardware implementations, this corresponds to the Register Transfer Level (RTL). One of the major countermeasures in this category is "masking" which consists in processing masked data rather than the data itself. The goal of the protection by masking is to randomize the power consumption and thus decorrelate as far as possible the computation activity from the secret information, which is generally the key for cryptographic algorithms. For this reason, masking provides a constant power consumption mean. As explained in detail in the next section, the mask has to be saved or processed.

Another means of providing protection consists in obfuscating the computation by generating noise which decreases the Signal to Noise Ratio "SNR" or more precisely the "leakage" to noise ratio. This makes the secret signal indiscernible. For instance, extra glitches in combinational gates can be added or extra jitter noise inserted at the clock stage. In software this can lead to the insertion of dummy instructions. Theoretically, this type of countermeasure is not very robust as the adversary can increase SNR by using more traces. For instance, the extra noise generated by the increase in pipelining reported in [53] or by unrolling the implementation as reported in [5] do not provide efficient protection against SCA.

At both architecture and netlist levels, one of the most efficient protection technique relies on the use of a differential logic. The efficiency of this type of countermeasure, often called "hiding", is based on attempting to make the power consumption constant by using dual-rail logic split in `True` and `False` networks. The rationale is to have one network consume power, while the other does not. One advantage of the hiding technique is to provide natural protection against fault attacks as explained in Sect. 4.4.

In this chapter we focus on two major countermeasures, masking and hiding. In particular, we discuss various differential logic styles that are well suited for FPGAs.

4.3 Countermeasures Based on Masking

4.3.1 Masking Principle

The masking countermeasure is certainly less complex to implement in FPGAs than elsewhere as it is applied at the architectural level only. Masking is performed on internal variables that are transformed into shares of masked variables and the mask itself. Software and hardware implementations both take advantage of this countermeasure, which has been the subject of many studies [2, 8, 19, 34]. The masking technique relies on concealing internal sensitive variables x by a mask m which takes random values. The internal variable x does not exist as a net in the cryptosystem but can be reconstructed by a pair of signals $(m, x_m = x\ \theta\ m)$, where x_m is the masked variable and θ is an operation which can be Boolean or arithmetic. Boolean masking uses the bit wise exclusive-or (xor) operation:

$$x_m = x \oplus m,$$

whereas arithmetic masking typically uses a modulus operation on a finite field:

$$x_m = x + m \pmod{n} \quad \text{or}$$
$$x_m = x * m \pmod{n},$$

where $n = 2^{|x|} = 2^{|m|}$ is equal to the number of values of the sensitive value or of the mask.

Indeed, for a correct masking scheme, the mask (and therefore the masked data) must be uniformly distributed throughout the secure data flow.

Another way to use the mask is the "random pre-charging" method [7]. This consists in temporal stages alternating between the mask m and the internal variable x. As a result, power consumption, which is mainly caused by the Hamming distance of two consecutive values, is not directly correlated with x. The drawback in hardware implementation is a decrease in throughput which is approximately a factor of 2 compared to "spatial" masking.

The implementation of masking is simple when the function f has the following linearity property:

$$f(x\theta m) = f(x)\theta f(m),$$

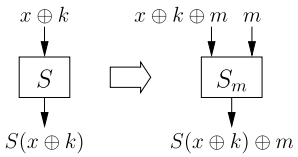
where θ is still a group operation.

The value of $f(x)$ can be reconstructed from the application of $f(x\theta m)$ and $f(m)^{-1}$, hence the computation of $f(x)$ can be extracted at the very end of the algorithm. This avoids direct leakage of information as $x\theta m$ and m are independent from x (as in Shannon's notion of "one-time pad" [50]).

If f is non-linear, the masking structure becomes more complex as $f(x)$ cannot be reconstructed mathematically from $f(x\theta m)$ and $f(m)$.[1] In symmetrical ciphering algorithms, the non-linear part corresponds to the S-boxes S. A common technique applied in software is to use a specific memory acting as a LUT S_m such that

[1] At least, not in a straightforward manner, *i.e.* without inverting f if it is invertible.

Fig. 4.2 S-box for masking



$S_m(x \oplus m) = S(x) \oplus m$. Consequently the size of this memory to implement the new table increases from 2^n to 2^{2n}, n being the number of bits of the mask.

Figure 4.2 illustrates the complexity change when this masking scheme is used.

It should be noted that it is not secure in hardware, because the register transfers unmask the data. The leakage, in the Hamming distance [53] model (which is implicit in FPGAs), is expressed as:

$$\underbrace{x \oplus m}_{\text{initial value}} \oplus \underbrace{S(x) \oplus m}_{\text{final value}} = x \oplus S(m).$$

For AES, masking can take advantage of the fact that the S-box is a combination of the inverse function in GF(2^8) and an affine function as proposed in [2] and [61]. However this implementation is very sensitive to zero-value attack [18]. This attack can be prevented by using the implementation proposed by Oswald in [39], which takes advantage of the multiplicative masking in GF(4) with only a slight increase in complexity.

4.3.2 Masking Implementations and Vulnerabilities

The robustness of masking countermeasures based on Boolean operators are provable against first order attacks [6], a first order attack being an attack where only the variable x is considered. However masking logic is sensitive to Higher-Order Attacks (HO-DPA) [35], where the attacker uses multiple observations of the same sensitive variable or multiple correlated variables. HO-DPA efficiency is directly related to the knowledge of the leakage, the way to observe the correlated variables [55] and the complexity of the masking implementation [2, 39, 40]. A pitfall introduced by the masked logics is the leakage of sensitive information through glitches. Unless special care is taken, the glitches can indeed depend on unmasked sensitive information, since some gates are likely to combine the masked data with the mask, thereby generating temporarily unprotected transitions [30]. Some special gates [12, 17] or synthesis techniques [37, 38] have been proposed to counter this effect.

In this chapter we will explain the HO-DPA on the classical hardware masking implementation referred to as "zero-offset" [62] and is illustrated in Fig. 4.3.

In the "zero-offset" circuit the second-order DPA can be performed by using observations of both the masked data $x_m = x \oplus m$ and the mask m that are computed concurrently. In order to understand the second-order DPA principle, let us consider the PMF (Probability Mass Function) of the activity corresponding to those of

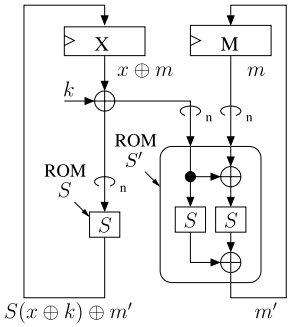


Fig. 4.3 "Zero-offset" masked DES, implemented with ROMs

the combined X and M registers in Fig. 4.3. The activity of these two registers is expressed by:

$$A = HW[\Delta(x,k) \oplus \Delta(m)] + HW[\Delta(m)]. \tag{4.1}$$

Where Δ expresses the distance of a register output, *i.e.*

$$\Delta(x,k) \doteq x \oplus S(x \oplus k),$$
$$\Delta(m) \doteq m \oplus m'.$$

If the registers have four bits (as for DES implementation), there are five possible PMFs depending on the $HW(\Delta(x,k))$ values, when the key is correct, as shown at the top of Fig. 4.4.

When the key is incorrect, the leakage corresponds to that of function A described in Equation 4.1 where:

- x is uniformly distributed in [`0x0`, `0xf`], because the guessed key is wrong,
- m is uniformly distributed in [`0x0`, `0xf`], because the mask is random and unknown to the attacker.

An HO-DPA attack of special interest is the mutual information analysis (MIA) introduced by Gierlichs [15].

It uses the Mutual Information $MI(O; \Delta(x,k) \oplus \Delta(m) + \Delta(m))$ as a distinguisher to build the attack, where optimized attacks close to MIA for example, variance based power attacks (VPA) [29] or entropy based power attacks (EPA) [28] can be devised.

4.3.3 Example of Protection for DES

Let us consider the "zero-offset" implementation of DES studied at UCL [54]. Its iterative architecture is illustrated in Fig. 4.5. This algorithmic masking associates a mask (ML, MR) with the plaintext (L, R).

At each round $i \in [0, 16[$, one intermediate mask (ML_i, MR_i) is calculated in parallel with the intermediate cipher word (L_i, R_i). If we leave aside the expansion

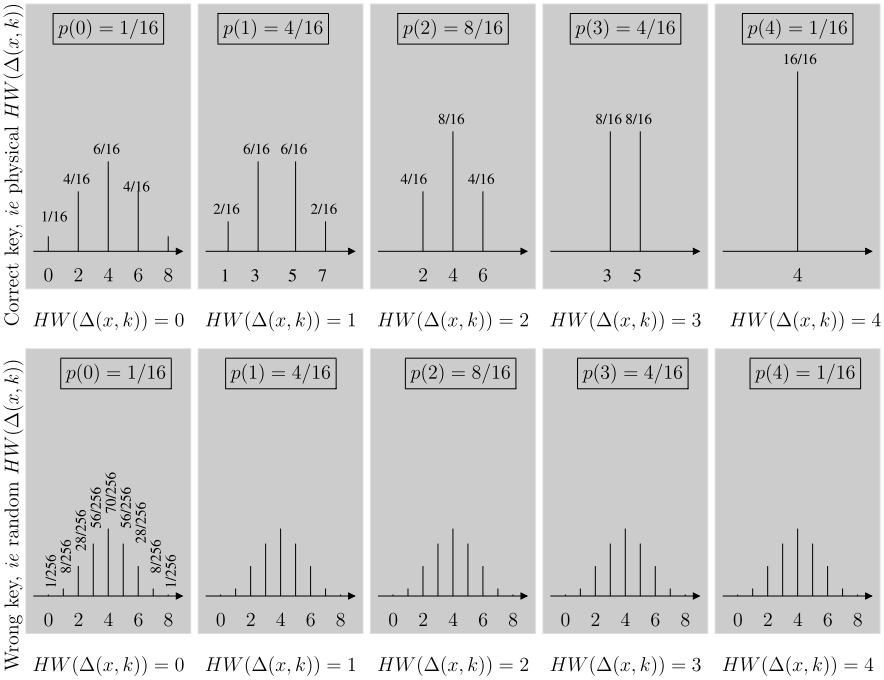


Fig. 4.4 PMFs corresponding to the five possible values of $HW(\Delta(x,k))$

E and the permutation P, the DES [1] round function f is implemented in a masked way by using a set of functions S and a set of functions S':

$$\begin{aligned} S(x_m \oplus k) &= S(x \oplus m \oplus k) = S(x \oplus k) \oplus m', \\ m' &= S'(x_m \oplus k, m) = S'(x \oplus m \oplus k, m). \end{aligned} \tag{4.2}$$

The variable m' is a new mask that can be used again in the next round.

The set of functions S contains the traditional S-boxes applied on masked intermediate words. The size of each S is 64 words of 4 bits when implemented with a ROM. The function S' is a new table which has a much greater ROM size of 4 K words of 4 bits, as there are two input words of 6 bits.

To mitigate the attacks on "zero-offset" implementations one possible solution is to balance the distribution. Authors in [27] propose to "squeeze" the leakage by inserting specific bijections before and after storing the mask. These bijections can be part of the "masked" ROM, see Fig. 4.6. The intermediate data (*e.g.* Sboxes output) have been protected by the same strategy, so as to ensure seamless "squeezing" throughout the combinational logic.

For AES, the implementation of the masked SubBytes structure S' would lead to ROM memory blocks with size 2^{16} words of 8 bits, which represents a huge increase in complexity. It is thus preferable to use substitution boxes with operators computing in smaller fields than GF(2^8) as in [39].

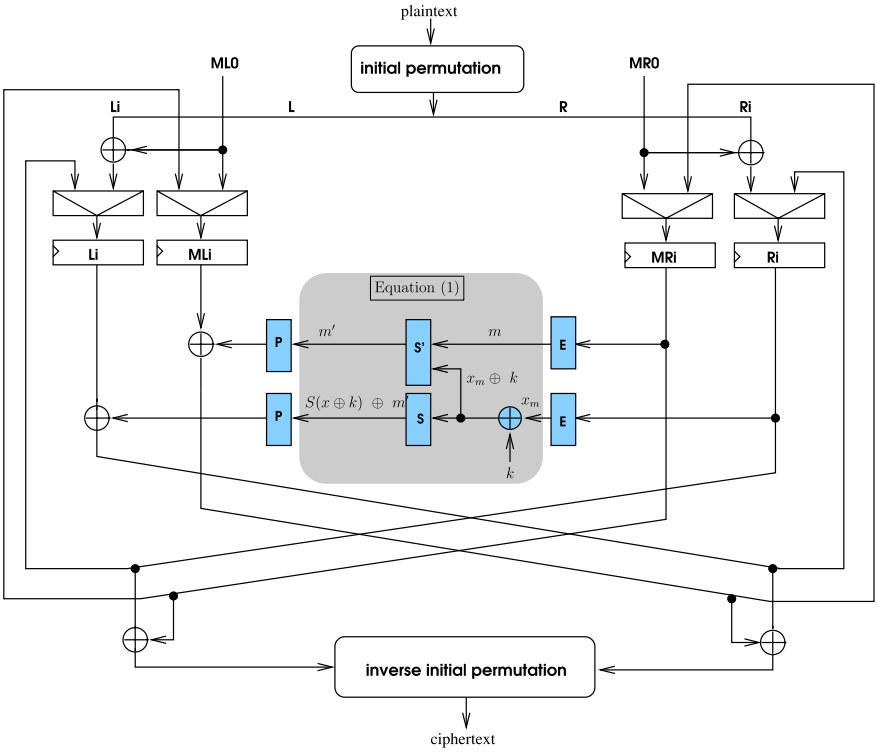


Fig. 4.5 Masked DES datapath

Another possible solution is reduce the implementation cost by regenerating m' from Eq. (4.2):

$$m' = S(x \oplus k \oplus m) \oplus S(x \oplus k). \tag{4.3}$$

This implementation called "universal S-Box masking" (USM) is illustrated in Fig. 4.7. It is such that some non-masked values (namely the S-box input and output) appear clearly in the implementation. This undoubtedly represents a potential threat. The size is merely increased by a factor of 2 due to the fact the S-box are duplicated. This is much less than the increase proposed in [54] where look up tables are 4 K $\times$ 4 memories.

In order to protect the USM implementation, the above mentioned squeezing leakage principle can be applied. The protected USM is shown in Fig. 4.8. It is composed of layers of the encoded table including input and output bijections. The bijections have to be chosen to ensure the maximum possible balance between the distribution of activity for the right key. For instance a robustness evaluation facilitates this choice. The bijections for the expansion and the permutation of DES are linear ("XOR with constant" operation) as these functions split the 32-bit words. However it is important to use non-linear bijections (at least 3 bits) for the S-Box ta-

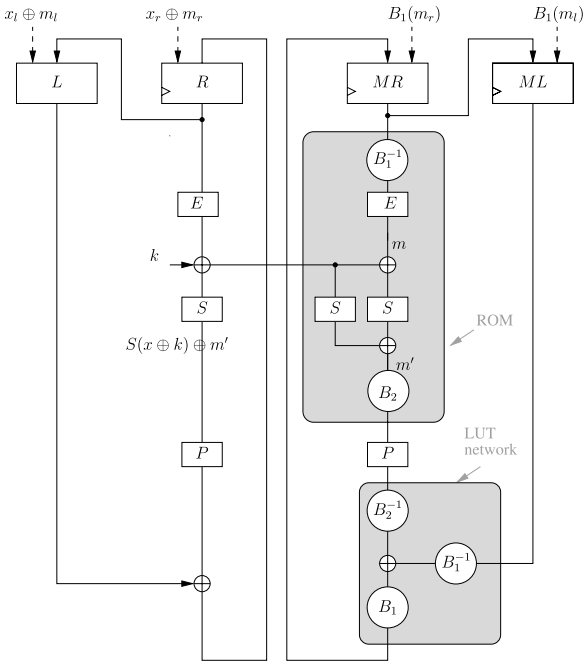


Fig. 4.6 Leakage squeezing of DES with a masked ROM implementation

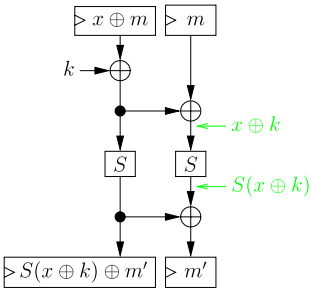


Fig. 4.7 Universal masked f function (of DES), with transiently unmasked values highlighted in *green* (color online)

ble as the Hamming distance reveals the unmasked value if linear bijection are used. The tables are implemented either in LUT networks or FPGA embedded RAMs.

These implementations were tested in a STRATIX II FPGA which is based on an adaptive LUT Module (ALM) cell. They were compared with unprotected DES, masked ROM and masked USM implementations without any leakage squeezing.

Table 4.1 summarizes the memories needed for each implementation and the estimated throughput.

These results show that in hardware implementations, the leakage squeezing method has little impact on complexity and speed. Moreover the USM implementation is particularly efficient as it avoids the use of large ROMs while maintaining high throughput.

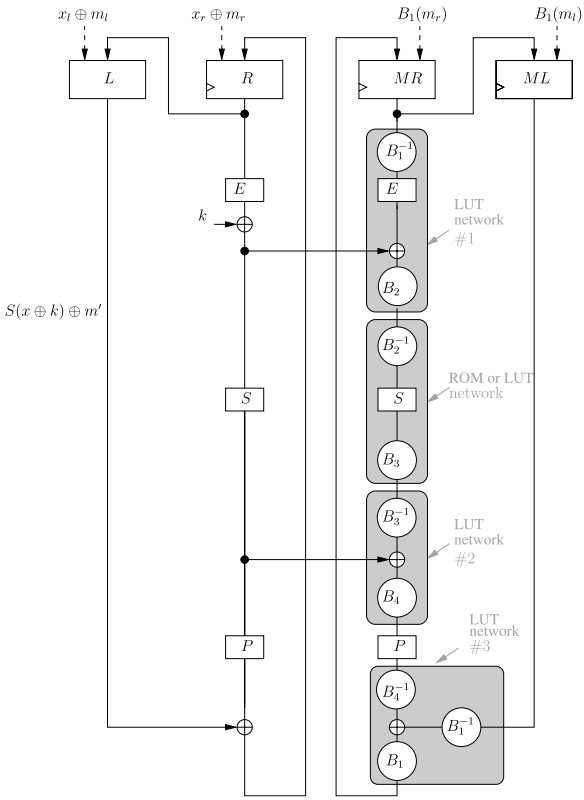


Fig. 4.8 Leakage squeezing of DES with a masked USM implementation

Robustness can be evaluated by using the theoretical framework introduced by Standaert et al. in [52]. The authors suggest analyzing side channel attacks with a combination of information theoretic and security metrics. These metrics aim at evaluating the amount of information provided by a leaking implementation and the possibility to turn this information into a successful key recovery.

Figure 4.9(a) shows the mutual information values obtained for each kind of implementation on simulated traces with respect to an increasing noise standard deviation over [0.1, 10] (*i.e.* an increasing SNR over [−20, 20]).

Table 4.1 Complexity and speed results. "l. s." denotes the "leakage squeezing" countermeasure

Implementation	ALMs	Block memory [bit]	M4Ks	Throughput [Mbit/s]
Unprotected DES (*reference*)	276	0	0	929.4
DES masked USM	447	0	0	689.1
DES masked ROM	366	131072	32	398.4
DES masked ROM with l. s.	408	131072	32	320.8
DES masked USM with l. s.	488	0	0	582.8

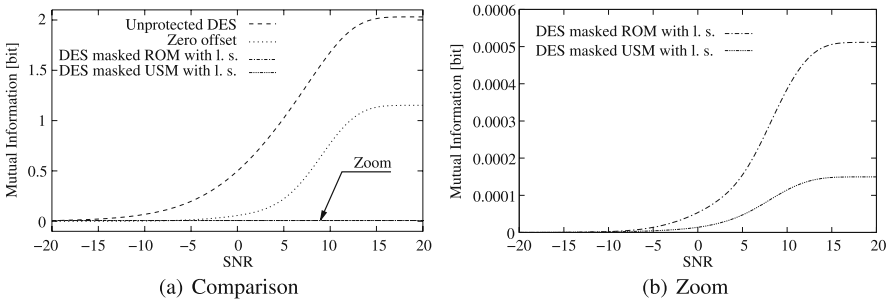


Fig. 4.9 Mutual information metric computed on several DES implementations

These results demonstrate the reduction in information leakage implied by the use of the leakage squeezing technique. As expected, the two implementations based on leakage squeezing leak less information than the zero offset implementation and the unprotected DES for all SNRs. Figure 4.9(b) is a zoom on the evolution of the mutual information in the case of the implementations based on the leakage squeezing technique in order to compare them.

Next real attacks were carried out on real power consumption traces in both the "zero offset" and "USM with squeezed leakage" implementations. For each scenario, a set of 25,000 power consumption traces was acquired using random masks and plaintexts. The first order success rate as described in [52] was calculated for different attack distinguishers; VPA [29], EPA [28] and MMIA [16]. The result is given in Fig. 4.10.

We can see that the attacks based on various distinguishers perform well in the "zero offset" implementation but not in the leakage squeezing implementation.

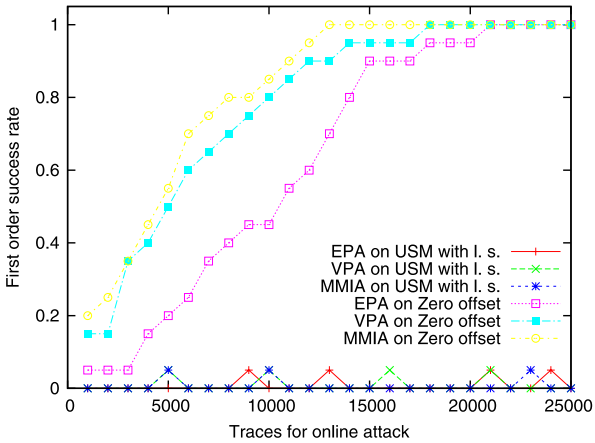


Fig. 4.10 First order success rate of 3 distinguishers, FPGA implementation

4.3.4 Example of Masked Processor for a Software Implementation

As explained in Chap. 2, experimental results suggest that pipelined processors increase the risk of SCAs, and have to be considered with care. A typical masking implementation for embedded processors in FPGAs requires some considerations.

One solution consists in implementing a dual pipelined datapath. Basically, the idea is to introduce a special datapath for the mask itself, which can be coupled to the classic RISC pipeline. Hence, instead of directly handling raw data, the processor operates on a dual datapath with masked data. The main role of the new datapath is to keep the corresponding mask for each masked data along the pipeline.

By far, the main difficulty is encountered during the EX stage, where all mathematical operations are implemented. As a first hypothesis, the ALU operations are not customized for any mask in order to compute the correct value. Also even if SCAs are still effective on combinational logic, one may consider that the leakage at the register stages is predominant. Indeed, to tackle clock skew issues, buffers are used to drive long lines, involving thus increased power consumption at the register level. This is especially the case for FPGAs. Hence, the ALU operations are performed with unmasked data, whereas the EX pipeline registers are masked.

Moreover, RISC-based architectures are structured around load-store instructions. All potential critical data coming from the data memory use load instructions and could take advantage of a masking scheme when going to the register banks. This approach not only offers the advantage to handle any instruction using a masking scheme but also provides a full compatibility with the processor's instruction set.

A MicroBlaze instruction set compliant processor, the SecretBlaze [3], has been developed for Xilinx FPGA. It implements the ideas of the RISC-based masked datapath. Among different types of masking, the boolean masking was chosen because of its low overhead cost and its good integration into the pipeline. Hence, the masked data result from XOR operations between the raw data and the mask values.

The SecretBlaze provides a dual datapath, two register files, a MAsked Memory Unit (MAMU), and a pseudo random generator (PRNG). The PRNG generates a 32-bit mask value at each clock cycle. The goal of the MAMU is to manage memory accesses with a static mask. Figure 4.11 illustrates the core block diagram of the SecretBlaze. The main differences from the original MicroBlaze are highlighted in gray.

The masking strategy is performed whenever a new value is loaded from the data memory. The loaded data is immediately XORed with the mask generated from the PRNG. The effect of the static mask is afterward removed. Both, the masked data and the mask itself, are stored respectively into the register file and the mask register file during the next clock cycle. As a consequence, no unprotected data is stored into register files of the processor. By analogy, store instructions follow the same scheme in reverse.

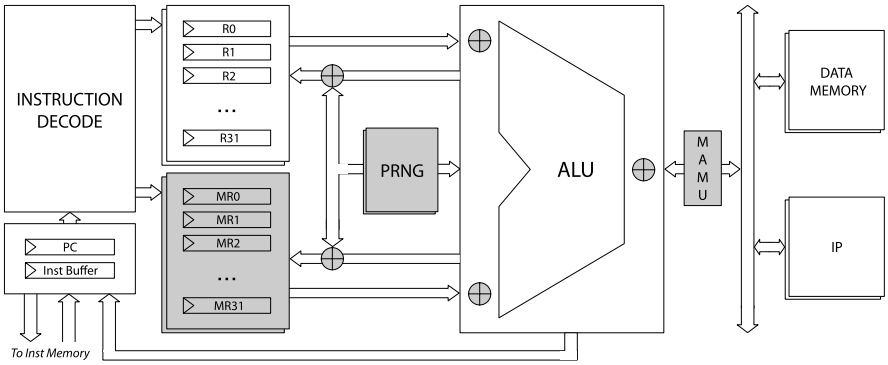


Fig. 4.11 SecretBlaze Block Diagram

For all instructions, the data have to be unmasked for the inputs of the ALU. Although another PRNG would have been the most secured solution, the mask of the first operand is XORed with the value of the PRNG in order to reduce the gate cost of the datapath (the XOR operation between two random numbers is a random number). The same remark can be made for the MA stage. Finally, instructions involving address computations such as loads, stores, as well as branches, have to be unmasked for the address assignments. This poses no significant threat because the addresses do not contain any sensitive information.

The performance and the resource impact of the proposed countermeasure were evaluated with an overhead of 80% of extra flip-flops, owing to the introduction of the PRNG and mask pipeline registers. Then, we observe a slight increase in the usage of slices and LUTs (+30%), related to extra-logic for the datapath of the mask. In terms of performance, the operating frequency is only reduced by 11.2%.

In order to evaluate the robustness of the masking technique, attacks were conducted on the processor running a software DES program, with and without the countermeasure at the hardware level. Results obtained show that the masked SecretBlaze offers a better resistance against DEMA of approximately a factor 2 during the execution of the critical instructions.

Figure 4.12 illustrates the DEMA obtained with 50,000 electromagnetic traces. Unlike the results observed in Chap. 2, this picture shows the efficiency of the protection, since the correlation at the different stages of the pipeline is no more relevant. However, the EX stage of the XOR instruction is still a weak point of the architecture. The conclusion to be drawn from these considerations is that the ALU, more generally combinational logic, is still a critical security issue in FPGAs, even registers are more numerous and more energy-consuming. Further investigations should be conducted to identify alternatives for securing the ALU of the processor, like hiding techniques detailed in the next section.

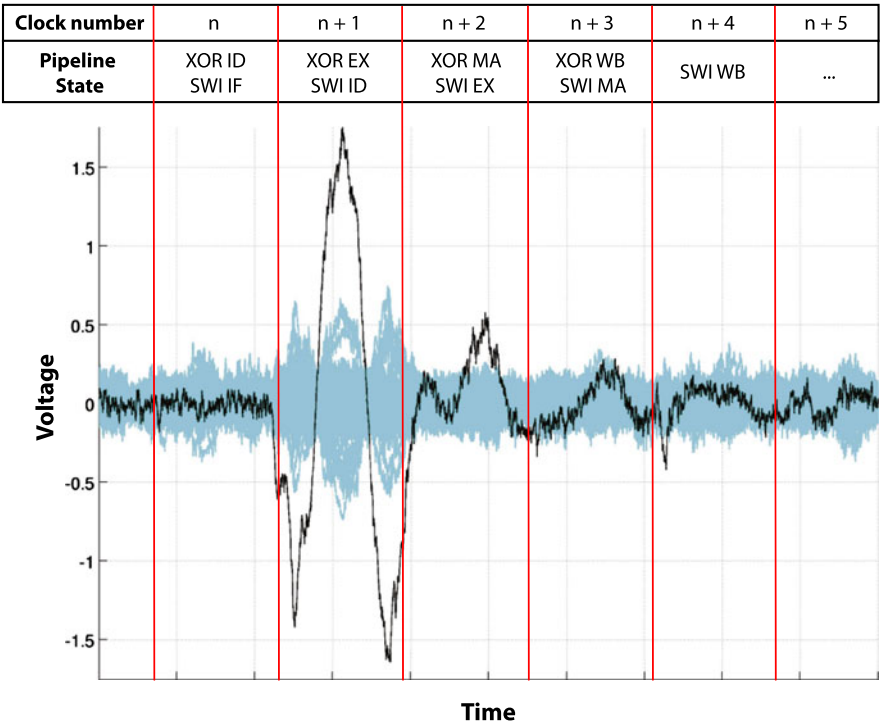


Fig. 4.12 DEMA traces obtained for the first sub-key of the DES software implementation with the SecretBlaze (*blue* = wrong key hypothesis, *black* = good key hypothesis) (color online)

4.4 Countermeasures Based on Hiding

4.4.1 Hiding Technique

The Hiding technique consists in achieving constant power consumption whereas Masking aims at averaging it. One way to obtain constant activity is to use differential logic characterized by the fact each variable is made up of two complementary signals. This logic is such that when one signal switches, the other does not and vice versa. This allows the design to be balanced in terms of activity since in CMOS technology the main power consumer is the switching rate. To make sure the number of transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) remains constant, the computation has two distinct stages:

1. A **Precharge** stage to reset all the signals in a known state, and
2. An **Evaluation** stage where the computation is performed with a fixed number of transitions.

The differential logic is also called "Dual Rail with Precharge Logic (DPL)" because the two signals from the same variable need twice as many routing resources. Therefore the complexity is at least twice that of an unprotected implementation.

The DPL signalization of the variable a is conveyed by two wires (a_t, a_f) for each Boolean variable, a_t is the TRUE signal and a_f is the FALSE. The state of the variable is either:

- NULL $= (0, 0)$ or $(1, 1)$ while in **Precharge**.
- VALID $\in \{(0, 1), (1, 0)\}$ while in **Evaluation**.

Therefore, every evaluation consists in the transition of exactly one wire ($(0, 0) \rightarrow (0, 1)$ or $(0, 0) \rightarrow (1, 0)$). If the design is properly balanced, which transition actually occurred is indiscernible by an attacker. The computation with DPL logics is a structure of a TRUE and FALSE networks with possible crossing wires interpreted as inversions. Although perfectly sound at logical level, in practice, DPL ends up being implemented in physical devices where the timing parameters impact the balance between the TRUE and FALSE networks. This unbalanced behaviors can damage the level of protection provided by DPL logics. Between the precharge to evaluation, and vice-versa, there may be:

1. Spurious transitions, referred to as glitches, that negate the hypothesis of activity invariability.
2. Early Evaluation (EE) effects. This takes place if the gate switching depends on the difference between the arrival time of the inputs.
3. Technological Bias (TB). This flaw results from the imbalance between the dual signals. It can be caused by manufacturing dispersion, by the place-and-route stage or merely by the types of gate driving the true and false networks. This could be exploited by an attacker who measures the signal from one wire of a pair.

In this section, we focus on the different styles that map very well in FPGAs: WDDL [58], STTL [43], BCDL [11, 36].

4.4.2 WDDL and Its Variants

The Wave Dynamic Differential Logic (WDDL) proposed by Tiri [58] is one of the simplest DPL logic styles.

The NULL state $(0, 0)$ is propagated by a wave of $(0, 0)$ pairs through the netlist thanks to the use of positive gates. A Boolean function f is said to be positive if for two Boolean variables x and y:

$$x \cdot y = x \quad \Rightarrow \quad f(y) \geq f(x).$$

This type of function corresponds to an assembly of AND and OR gates. Figure 4.13 illustrates a two-input "AND2" gate, the logic network TRUE T receives the two TRUE inputs a_t and b_t. The dual "OR" function is implemented by the

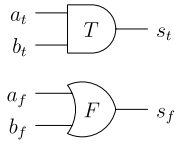


Fig. 4.13 "AND" WDDL gate

F network which receives the FALSE inputs a_f et b_f. Thus $T(x) \doteq H(x)$ and $F(x) \doteq \overline{H(\overline{x})}$, $F(x)$ can be obtained by using the De Morgan's law.

Non-positive logic like inverters or NAND gates are implemented by crossing the two networks. The number of Flip-Flops is necessarily multiplied by four. These are duplicated for the two TRUE and FALSE networks and also for the precharge and evaluation stage as shown in Fig. 4.14.

The timings in Fig. 4.15 show that the number of transitions is the same (two) during the Precharge ⇒ Evaluation stage and vice-versa. As in CMOS technologies, power consumption is directly correlated with this number, the logic is reputed to be balanced.

The positivity of WDDL ensures the absence of glitches in the complete netlist. However, as shown in [56, 57], WDDL is prone to early evaluation (EE). The early evaluation effect is due to the difference in timing between two variables of one gate. This timing difference is transferred to the WDDL gate output during the transitions Precharge ⇔ Evaluation. Figure 4.16(b) illustrates the principle of early evaluation for a 2-input AND gate and its dual 2-input OR gate, as represented in Fig. 4.13. Output switching Δt_1 is different from Δt_2 and therefore can reveal information about the state of the inputs.

In addition, WDDL still has technological bias (TB) *i.e.* imbalance between the dual TRUE and FALSE networks at both structure and routing levels. Constraining

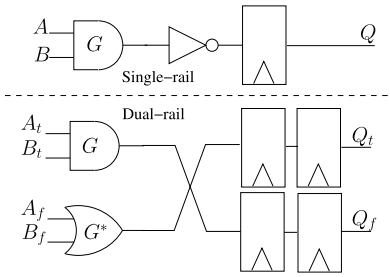


Fig. 4.14 A digital circuit and its WDDL equivalent

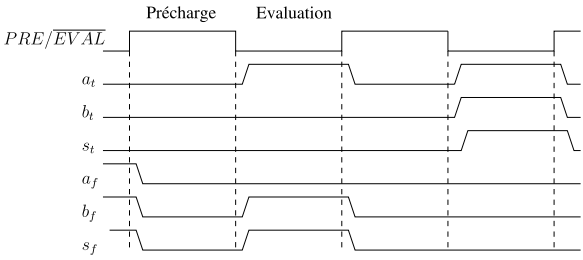


Fig. 4.15 Simulation showing the WDDL timings

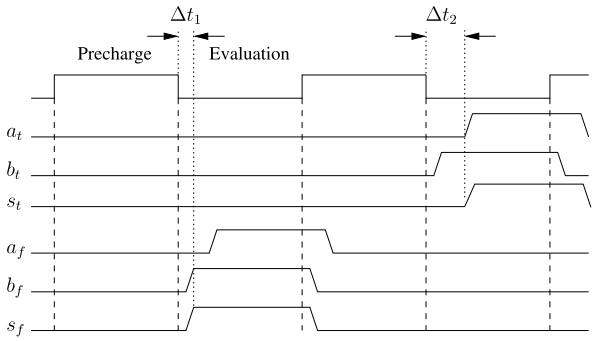


Fig. 4.16 Early evaluation

routing is not so easy in FPGAs because the interconnect structure is confidential. The two causes of imbalance plus the EE effect have enabled some attacks on WDDL circuits, as described by the authors of WDDL themselves in an ASIC [60] or independently in an FPGA [44].

The impact of the place and route (P/R) steps on the timings of dual rail designs is of major importance to obtain dual rail balance and thus reduce the correlation between the processed data and power consumption. Without any P/R constraints it was shown in [44] that WDDL is attackable. Novel P/R strategies that take advantage of FPGAs with cells composed of two-output LUTs have appeared. For instance in [45] the balancing strategy is to place and route the gate in the same STRATIX II ALM.

Some variants of WDDL have been devised to facilitate the balance of the WDDL networks. For instance Double WDDL (DWDDL) was introduced in [63] to counterbalance one unbalanced network with a dummy dual one. The main drawback of DWDDL is its complexity which is double that of WDDL.

Isolated WDDL (IWDDL) [31] is a different strategy to separate a WDDL netlist into two unconnected halves. Here, inverters are kept but a potential glitch is stopped by systematically inserting one register after it. This strategy is expensive in terms of gate complexity and requires a redesign of the controller. Additionally, the design becomes much more pipelined, which requires much higher clock frequencies to maintain an acceptable throughput. However, the advantage of this approach is stopping the propagation of the EE wave.

As DWDDL and IWDDL are complex but theoretically robust, one point is questionable: won't completely separating the netlist open the door to well located EMA attacks that can selectively record the activity of one specific part of the circuit, thus defeating the activity invariability property.

4.4.3 Synchronized Logics: STTL, BCDL

Another strategy to get rid of the early evaluation effect is to synchronize the variables before starting the gate evaluation and precharge stages. However, to make sure no glitches occur, the following conditions have to be met:

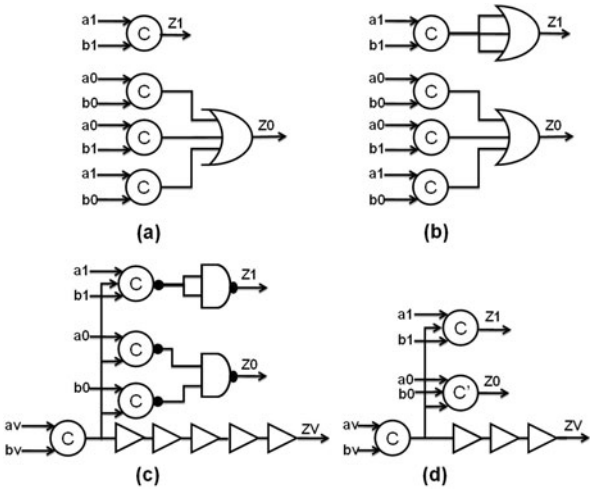


Fig. 4.17 Two-input AND gate implementations: (**a**) basic dual rail AND (**b**) SecLib dual rail AND, (**c**) STTL AND (**d**) compact STTL AND

1. Evaluation starts after all the input signals are valid.
2. Precharge starts by following one of these two rules:

   - Rule `sync-1`: Precharge starts after all the inputs becomes NULL.[2]
   - Rule `sync-2`: Precharge starts before the first input becomes NULL.

If the precharge is always late (rule `sync-1`), the gate outputs need to be memorized. While for rule `sync-2` no memorization is necessary but there is a specific precharge signal.

STTL In [43], the authors suggest using an additional third wire to synchronize the input arrivals by using C-elements to create the Secure Triple Track Logic (STTL), like in Asynchronous logic. In this case, rule `sync-1` applies. This third wire should indicate whether the output data is stable (and thus valid) or not. Figure 4.17 displays different implementations of a dual rail two-input AND gate.

Figure 4.17(a) represents the basic dual rail AND in asynchronous logic. Figure 4.17(b) is a more secure dual rail AND gate, also called SecLib [22] where the two dual outputs are balanced. Figures 4.17(c) and 4.17(d) represent the STTL AND gate, with a more compact triple rail AND in (d). Operator C stand for a C-element [49], ($Z = (a+b)\cdot c + Z\cdot(a+b+c)$), and C' for a generalized C-element. Implementations (b), (c) and (d) are power balanced. However, the third rail in (c) and (d) must fulfill a timing constraint to effectively obtain a quasi data independent timing behavior at block level.

The validity output pin ZV of triple rail gates is controlled by buffers, three in the case of Fig. 4.17(d). These buffers ensure that the delay Θv in propagation from the validity inputs (av, bv) to the output ZV remains greater than the delays Θd from $(a1, a0, b1, b0)$ inputs to the data outputs $(Z0, Z1)$. Note that the number of

[2]NULL is the value in precharge phase.

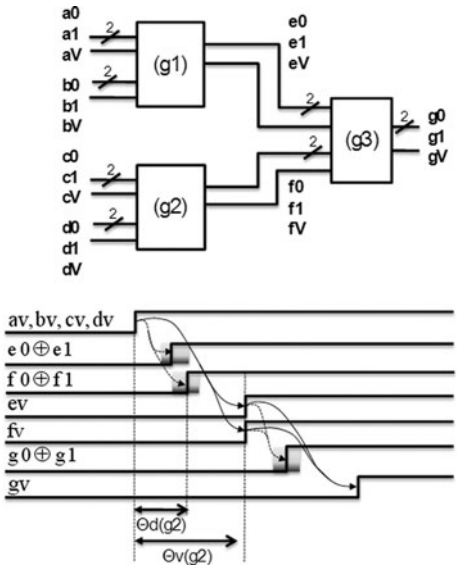


Fig. 4.18 The basic operation of secure triple track logic

buffers must be defined by the designers to guarantee that this timing characteristic is satisfied even in the presence of output load mismatches introduced by the place and route step as described in [43]. With such design guidelines of triple rail gates, we can confidently guarantee that the time at which a triple rail gate fires is independent of the specific data processed by its containing block.

Figure 4.18 illustrates this key characteristic of secure triple rail logic. After the firings of av, bv, cv and dv (assumed to occur at the same time without loss of generality), $e0$, $e1$, $f0$ and $f1$ fire first. Then, the firing of ev and fv occur, which in turn triggers $g0$ or $g1$, followed by gv, since validity rails have a longer propagation delay. Thus the firing of triple rail gates is triggered by the validity rails characterized by a switching speed lower than that of data rails. In other words, the validity rail array (arrows in Fig. 4.18) operates as a backbone of the logical block, sequencing the events independently of data processing (dashed arrows in Fig. 4.18).

During the firing sequence, the time at which $e0$ ($f0$, $g0$), $e1$ ($f1$, $g1$) settle may diff, due to possible output load mismatches. This is represented by the gray rectangles on Fig. 4.18. However, these arrival time mismatches do not affect the firing of the following gates, which are triggered by the validity rails. This characteristic avoids the effect of load mismatches piling up on timing along data paths. This guarantees quasi data independent power consumption and computation time at the block level.

BCDL Balanced Cell-based Differential Logic (BCDL) [11, 36] is a synchronized logic which takes advantage of a global signal that allows the designer to both reduce the complexity of the design and speed up the calculation. This global precharge signal called *PRE* allows rule `sync-2` to be fulfilled, and as a result, no memorization function, such as C-elements, is needed.

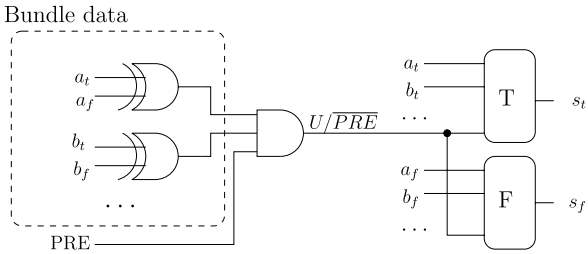


Fig. 4.19 The basic BCDL cell

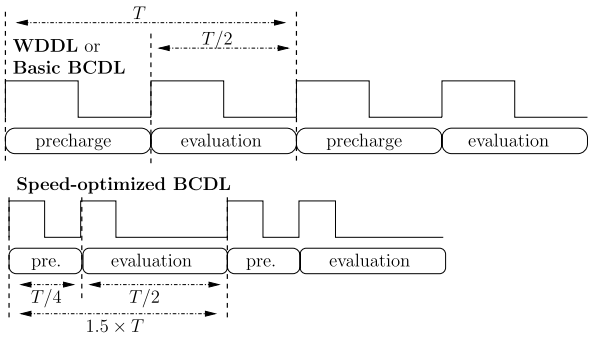


Fig. 4.20 Timing optimization in the DPL protocol when the precharge time is reduced

The basic BCDL gate is presented in Fig. 4.19. Synchronization is performed by a "unanimity to 1" operator on the left side of the figure and can operate on a bundle of data.

The global *PRE* signal is constrained to be faster than any inputs. Consequently when the $U/\overline{PRE}$ of Fig. 4.19 falls to 0 $\Rightarrow$ the precharge is forced, and when $U/\overline{PRE}$ rises to 1 $\Rightarrow$ the evaluation begins after "unanimity to 1".

As the calculation in Tables T and F can be completely separated, the complexity of the tables is reduced as they receive 2^{n+2} inputs rather than 2^{2n} in others DPL, n being the number of gate inputs. Therefore the implementation with embedded RAM in FPGAs is appropriate. Another complexity gain is the ease of implementation of 2-input gates as they are not limited by the positiveness constraint. For instance, a XOR BCDL gate including synchronization can be done in one ALM of STRATIX II or one LUT6 (dual LUT5) in VIRTEX 5 families.

As the BCDL design allows squeezing of the precharge step, it can compute about 75% faster than WDDL because the precharge is global. This possibility is depicted in Fig. 4.20.

Table 4.2 gives the complexity and speed of an AES implementation in a STRATIX FPGA for WDDL and BCDL.

Table 4.2 Complexity and speed of an AES implementation in WDDL and BCDL

	ALM	Reg	RAM	Max. freq.	Max. throughput
No protection	1078	256	40 Kb	71.88 MHz	287.52 Mbps
WDDL	4885	1024	–	37.07 MHz	74.14 Mbps
BCDL	1841	1024	160 Kb	50.64 MHz	151.92 Mbps

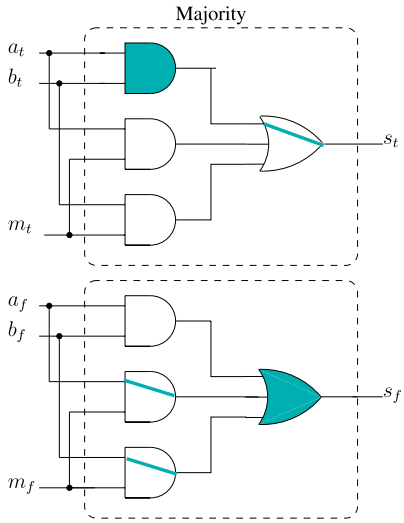


Fig. 4.21 MDPL AND gate

4.4.4 DPL with Masking: MDPL

Masked Dual-rail with Precharge Logic (MDPL [41]) is an attempt to fix the otherwise imbalance of WDDL. The assumption is that, in some conditions, it may be difficult to constrain a router to balance the differential interconnect. Indeed, the two solutions available in the literature, namely the fat wire [59] and the backend duplication [20] methods, apply primarily to ASICs. Transposition to FPGA is possible, although with less fine grained control over the result [21]. For this reason, MDPL swaps the true and the false routes with a random mask, so as to protect from fatal routing unbalance. By the same token, it makes up for the structural unbalance of the dual pair of gates. The only gates involved in the logic are majority functions, both for the true and the false networks. Figure 4.21 represents the MDPL AND gate. When the random mask m is 0 the `TRUE` network performs the AND, whereas the `FALSE` network performs the OR, which is the dual gate of AND, it is the other way round when m is 1.

Although MDPL fails to provide a solution to the early evaluation and precharge of WDDL as presented in [42], it could be efficient against the TB effect. It has been shown that the mask itself could be attacked [46]. Consequently MDPL was enhanced and became the improved MDPL (iMDPL) by adding a synchronization

stage [24]. The Dual Rail switching Logic (DRSL) [10] is very similar to iMDPL and more dedicated to ASIC. However it has been shown in [11] that without P/R care the DRSL can generate glitches when returning to the precharge phase. Synchronized logic like BCDL can take advantage of the random masking to mitigate the TB if few constraints apply during the P/R phase. For instance, BCDL with a mask (MBCDL) would need an extra mask input to the evaluation tables for random swapping of the `TRUE` and `FALSE` networks.

4.4.5 Intrinsic Fault Resilience of Dual-Rail With Precharge Logics

Single bit faults are inefficient against DPL because they turn a VALID data into a NULL token, that propagates and results in a non-exploitable error since it hides the faulted value. This is the typical scenario described in the seminal paper [48], introducing the intrinsic immunity of DPL against some classes of DFA.

Highly multiple faults ($(1,0) \leftrightarrow (0,1)$) randomly generate a large quantity of NULL values along with some more unlikely but devastating bit-flips. However, as NULL values are systematically propagated, they proliferate very quickly after some combinatorial logic layers traversal. And as they have the nice property of being able to contaminate VALID values, the risky coherent bit-flips (simultaneous $0 \xrightarrow{*} 1$ and $1 \xrightarrow{*} 0$ in one dual-rail couple) is very likely to be jammed through the propagation towards the algorithm output. This absorption property is all the more efficient as the number of NULL generated by the multiple faults is high. Therefore, the only way to inject a 'poisonous' fault is to stress the circuit sufficiently to generate multiple faults, without nonetheless creating too many faults so as to leave a chance for them not to be absorbed during their percolation towards the outputs [4, 23].

4.4.6 Comparison of DPL Families

Table 4.3 compares robustness, speed and complexity of a few DPLs. In fact, it is difficult to evaluate robustness fairly, as the DPL countermeasure depends on the target technology and the quality of the P/R stages. However most non-synchronized DPL, such as WDDL and MDPL have been attacked without any particular P/R effort [42, 44, 57]. The analysis of DPL robustness is still an ongoing research project. Some ideas come from the information theory with mutual information analysis [15] and the stochastic approach [47]. In Table 4.3, robustness against SCA is indicated by the logic capacity to be insensitive to early evaluation and technological bias. The fault column indicates if the logic able to cope with symmetric fault (faults being '1' or '0'), which is preferable, rather than asymmetric. Fault detection can be combinatorial or sequential. If it is sequential the cost is higher.

Table 4.3 Comparison of robustness, complexity and speed of a few DPLs

Logic	Compl.	Speed	Robust. SCA		Robust. FA		Design Constr.
			EE	T. B.	Fault	Det.	
WDDL	*	$< 1/2$			asym	comb	Positive gates
MDPL	*	$< 1/2$		✓	asym	comb	MAJ gate + RNG
STTL	*	$< 1/4$	✓		sym	seq	50% more wiring
DRSL	*	$< 1/2$	partly	✓	sym	comb	+ RNG
IWDDL		$< 1/2 \cdot n$	✓		asym	comb	superpipeline
BCDL	**	$> 1/2$	✓		sym	comb	
MBCDL	*	$> 1/2$	✓	✓	sym	comb	+ RNG

Table 4.4 Hardware countermeasures overhead

Countermeasure	None: reference	Masking	Hiding: WDDL	Hiding: BCDL
Period	1	$\approx 1\times$	$\approx 2\times$	$\in [1, 2]\times$
Area: gates	1	$\approx 2\times$	$\approx 2\times$	$\approx 2\times$
Area: memory	$2^n \times m$	$2^{2n} \times m$	$2^{2n} \times 2m$	$2^{n+1} \times 2m$

Two stars in the complexity column means that the DPL needs less gates/memories to be implemented. The ratio with an unprotected implementation is given in the speed column. If the ratio is greater than $1/2$, this means the DPL has an accelerated precharge stage, like for BCDL. Finally the design constraints are listed, for instance the needs of an RNG.

4.5 Comparison of FPGA, ASIC and Software Countermeasures

As shown in 4.3 and 4.4, FPGAs and ASICs allow for the implementation of masking and hiding techniques that do not affect the processing speed too much. Those targets are indeed much easier to protect than software targets, since the designer has full control over implementation. Instead, on a processor, only some instructions are available, which makes the implementation of countermeasures very awkward.

The overhead of hardware implementations is given in Table 4.4. Regarding hardware countermeasures, it is remarkable that the throughput is almost unchanged. Indeed, the mask can be processed in parallel, the only interaction between the several shares occurring during recombination, merely consists in xor operations. In dual-rail logics, the precharge inevitably causes a dead cycle for each evaluation cycle. And since, in addition, some gates cannot be used for all logics (for instance, positive gates are required in WDDL), the complexity and speed results are worse than without protection. Thus, the throughput is about halved in WDDL. BCDL has the special feature that the precharge cycle can be shrunk, which results in an overhead in terms of throughput that is less than a factor of two. Regarding the number of

resources required to implement the countermeasures, they are about doubled, since two paths are created both for masking and hiding strategies. The biggest difference comes from the RAMs. In masking, unless special structures can be used (such as the factorizing of the S-Box in AES), the number of address bits for the masking is doubled. The same goes for the hiding style, where the number of output bits is also doubled. BCDL is an exception, as the precharge is global. Thus only one additional input bit is required (for the precharge global line) to zero the dual-rail output.

The difference between FPGAs and ASICs is due to the fact that RAMs are available in larger quantities in FPGAs. Thus makes all the countermeasures presented in this section especially attractive. In addition, the FPGAs can take advantage of their reconfigurability capability to mutate their implementation, thereby preventing some attacks that rely on the hypothesis of constant architecture.

Comparing the overhead of Hardware versus Software implementations is more difficult. First of all, no software "hiding" countermeasure has been proposed so far. Regarding masking, it is usually accepted that an overhead of 100 for the throughput is a reasonable approximation. The advantage of hardware is thus clear.

4.6 Conclusions

Cryptographic algorithms can be mapped without difficulty in FPGAs. Although these targets are *a priori* leaking more than ASICs, thanks to their genericity, they also welcome traditional countermeasures, typically those based on masking and hiding. Programming these countermeasures was shown to be feasible, and a number of case-studies have been cited. We conclude that the remarkable property of countermeasures in hardware is that they have almost no affect on the throughput of the algorithm. This contrasts greatly with software countermeasures, which are considerably slowed down when a rigorous masking is enforced. Also, the reconfigurability of FPGAs enables algorithmic countermeasures, such as period implementation update, which impedes attacks that rely on a stable side-channel measurement. This potentiality is rarely addressed in the literature; it is clear that such high-level countermeasures can be further enhanced for greater system-level security.

References

1. NIST/ITL/CSD. Data Encryption Standard. Fips Pub 46-3. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf (1999)
2. Akkar, M.-L., Giraud, C.: An implementation of DES and AES secure against some attacks. In: Proceedings of CHES'01, Paris, France, May. LNCS, vol. 2162, pp. 309–318. Springer, Berlin (2001)
3. Barthe, L., Benoit, P., Torres, L.: Investigation of a masking countermeasure against side-channel attacks for RISC-based processor architectures. In: FPL, pp. 139–144 (2010)
4. Bhasin, S., Danger, J.-L., Flament, F., Graba, T., Guilley, S., Mathieu, Y., Nassar, M., Sauvage, L., Selmane, N.: Combined SCA and DFA countermeasures integrable in a FPGA design flow. In: ReConFig, Cancún, Quintana Roo, México, December 9–11,

pp. 213–218. IEEE Comput. Soc., Los Alamitos (2009). doi:10.1109/ReConFig.2009.50. http://hal.archives-ouvertes.fr/hal-00411843/en/

5. Bhasin, S., Guilley, S., Sauvage, L., Danger, J.-L.: Unrolling cryptographic circuits: a simple countermeasure against side-channel attacks. In: RSA Cryptographers' Track, CT-RSA, San Francisco, CA, USA, March 1–5. LNCS, vol. 5985, pp. 195–207. Springer, Berlin (2010). doi:10.1007/978-3-642-11925-5_14
6. Blomer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. In: Proceedings of SAC'04, Waterloo, Canada, August. LNCS, vol. 3357, pp. 69–83. Springer, Berlin (2004)
7. Bucci, M., Guglielmo, M., Luzzi, R., Trifiletti, A.: A power consumption randomization countermeasure for DPA-resistant cryptographic processors. In: PATMOS. LNCS, vol. 3254, pp. 480–490. Springer, Berlin (2004)
8. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: CRYPTO, August. LNCS, vol. 1666 (1999). ISBN 3-540-66347-9
9. Chaudhuri, S., Danger, J.-L., Guilley, S., Hoogvorst, P.: FASE: an open run-time reconfigurable FPGA architecture for tamper-resistant and secure embedded systems. In: IEEE 3rd International Conference on Reconfigurable Computing and FPGAs (ReConFig 2006), San Luis Potosi, Mexico, September, pp. 1–9 (2006)
10. Chen, Z., Zhou, Y.: Dual-rail random switching logic: a countermeasure to reduce side channel leakage. In: CHES, Yokohama, Japan. LNCS, vol. 4249, pp. 242–254. Springer, Berlin (2006)
11. Danger, J.-L., Guilley, S., Bhasin, S., Nassar, M.: Overview of dual rail with precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors.—New attacks and improved counter-measures—. In: SCS, IEEE, Jerba, Tunisia, November 6–8, pp. 1–8 (2009). doi:10.1109/ICSCS.2009.5412599. Complete version online: http://hal.archives-ouvertes.fr/hal-00431261/en/
12. Fischer, W., Gammel, B.M.: Masking at gate level in the presence of glitches. In: CHES, Edinburgh, UK, August 29–September 1. LNCS, vol. 3659, pp. 187–200. Springer, Berlin (2005)
13. Garcia, A.: Power consumption and optimization in field programmable gate arrays. PhD thesis (in French). Ecole Nationale Supérieure des Télécommunications (August 2000)
14. García, A.D., Burleson, W.P., Danger, J.-L.: Power modelling in field programmable gate arrays (FPGA). In: FPL, Glasgow, UK, August 30–September 1. LNCS, vol. 1673, pp. 396–404. Springer, Berlin (1999)
15. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: CHES, 10th International Workshop, Washington, D.C., USA, August 10–13. LNCS, vol. 5154, pp. 426–442. Springer, Berlin (2008)
16. Gierlichs, B., Batina, L., Preneel, B., Verbauwhede, I.: Revisiting higher-order DPA attacks: multivariate mutual information analysis. In: CT-RSA, San Francisco, CA, USA, March 1–5. LNCS, vol. 5985, pp. 221–234. Springer, Berlin (2010)
17. Golic, J.D., Menicocci, R.: Universal masking on logic gate level. Electron. Lett. **40**(9), 526–528 (2004). doi:10.1049/el:20040385
18. Golic, J., Tymen, C.: Multiplicative masking and power analysis of AES. In: CHES, San Francisco, USA, vol. 2523, pp. 198–212. Springer, Berlin (2003)
19. Goubin, L., Patarin, J.: DES and differential power analysis. In: CHES, August. LNCS, pp. 158–172. Springer, Berlin (1999)
20. Guilley, S., Hoogvorst, P., Mathieu, Y., Pacalet, R.P.: The "backend duplication" method. In: CHES, Edinburgh, Scotland, UK, August 29th–September 1st. LNCS, vol. 3659, pp. 383–397. Springer, Berlin (2005)
21. Guilley, S., Chaudhuri, S., Sauvage, L., Graba, T., Danger, J.-L., Hoogvorst, P., Vong, V.-N., Nassar, M.: Place-and-route impact on the security of DPL designs in FPGAs. In: HOST (Hardware Oriented Security and Trust), IEEE, Anaheim, CA, USA, June, pp. 29–35 (2008)
22. Guilley, S., Chaudhuri, S., Sauvage, L., Hoogvorst, P., Pacalet, R., Bertoni, G.M.: Security evaluation of WDDL and SecLib countermeasures against power attacks. IEEE Trans. Comput. **57**(11), 1482–1497 (2008)

23. Guilley, S., Sauvage, L., Danger, J.-L., Selmane, N.: Fault injection resilience. In: FDTC, Santa Barbara, CA, USA, August 21. IEEE Comput. Soc., Los Alamitos (2010). Complete version: http://hal.archives-ouvertes.fr/hal-00482194/en/
24. Kirschbaum, M., Popp, T.: Evaluation of a DPA-resistant prototype chip. In: ACSAC, Honolulu, Hawaii, July, pp. 43–50. IEEE Comput. Soc., Los Alamitos (2009)
25. Kocher, P.C.: Leak-resistant cryptographic indexed key update, March 25, 2003. United States Patent 6,539,092 filed on July 2nd, 1999 at San Francisco, CA, USA
26. Kuon, I., Rose, J.: Measuring the gap between FPGAs and ASICs. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **26**(2), 203–215 (2007)
27. Maghrebi, H., Guilley, S., Danger, J.-L.: Leakage squeezing countermeasure against high-order attacks. In: WISTP, Heraklion, June (2011)
28. Maghrebi, H., Guilley, S., Danger, J.-L., Flament, F.: Entropy-based power attack. In: HOST, Anaheim Convention Center, Anaheim, CA, USA, pp. 1–6. IEEE Comput. Soc., Los Alamitos (2010). doi:10.1109/HST.2010.5513124
29. Maghrebi, H., Danger, J.-L., Flament, F., Guilley, S.: Evaluation of countermeasures implementation based on boolean masking to thwart first and second order side-channel attacks. In: SCS, IEEE, Jerba, Tunisia, pp. 1–6 (2009). doi:10.1109/ICSCS.2009.5412597. Complete version online: http://hal.archives-ouvertes.fr/hal-00425523/en/
30. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: CT-RSA, San Francisco, CA, USA. LNCS, vol. 3376, pp. 351–365. Springer, Berlin (2005)
31. McEvoy, R.P., Murphy, C.C., Marnane, W.P., Tunstall, M.: Isolated WDDL: a hiding countermeasure for differential power analysis on FPGAs. ACM Trans. Reconfigurable Technol. Syst. **2**(1), 1–23 (2009). doi:10.1145/1502781.1502784
32. Medwed, M., Standaert, F.-X., Groéschédl, J., Regazzoni, F.: Fresh re-keying: security against side-channel and fault attacks for low-cost devices. In: AFRICACRYPT, Stellenbosch, South Africa, May 03–06. LNCS, vol. 6055, pp. 279–296. Springer, Berlin (2010). doi:10.1007/978-3-642-12678-9_17
33. Mentens, N., Gierlichs, B., Verbauwhede, I.: Power and fault analysis resistance in hardware through dynamic reconfiguration. In: CHES, Washington, D.C., USA, August 10–13. LNCS, vol. 5154, pp. 346–362. Springer, Berlin (2008)
34. Messerges, T.S.: Securing the AES finalists against power analysis attacks. In: Fast Software Encryption'00, New York, April 2000 (2000)
35. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: CHES, Worcester, MA, USA, August 17–18. LNCS, vol. 1965, pp. 71–77. Springer, Berlin (2000)
36. Nassar, M., Bhasin, S., Danger, J.-L., Duc, G., Guilley, S.: BCDL: A high performance balanced DPL with global precharge and without early-evaluation. In: DATE'10, Dresden, Germany, March 8–12, pp. 849–854. IEEE Comput. Soc., Los Alamitos (2010)
37. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: ICICS, Raleigh, NC, USA, December 4–7. LNCS, vol. 4307, pp. 529–545. Springer, Berlin (2006)
38. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of non-linear functions in the presence of glitches. In: ICISC, Seoul, Korea. LNCS, vol. 5461, pp. 218–234. Springer, Berlin (2008)
39. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A side-channel analysis resistant description of the AES S-box. In: Proceedings of FSE'05, Paris, France, February. LNCS, vol. 3557, pp. 413–423. Springer, Berlin (2005)
40. Peeters, E., Standaert, F.X., Donckers, N., Quisquater, J.J.: Improved higher-order side-channel attacks with FPGA experiments. In: CHES. LNCS, vol. 3659, pp. 309–323. Springer, Berlin (2005)
41. Popp, T., Mangard, S.: Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In: Proceedings of CHES'05, Edinburgh, Scotland, UK, September. LNCS, vol. 3659, pp. 172–186. Springer, Berlin (2005)

42. Popp, T., Kirschbaum, M., Zefferer, T., Mangard, S.: Evaluation of the masked logic style MDPL on a prototype chip. In: CHES, Vienna, Austria, September. LNCS, vol. 4727, pp. 81–94. Springer, Berlin (2007)
43. Razafindraibe, A., Robert, M., Maurine, P.: Analysis and improvement of dual rail logic as a countermeasure against DPA. In: PATMOS, Göteborg, Sweden, pp. 340–351 (2007)
44. Sauvage, L., Guilley, S., Danger, J.-L., Mathieu, Y., Nassar, M.: Successful attack on an FPGA-based WDDL DES cryptoprocessor without place and route constraints. In: DATE, Nice, France, April, pp. 640–645. IEEE Comput. Soc., Los Alamitos (2009)
45. Sauvage, L., Nassar, M., Guilley, S., Flament, F., Danger, J.-L., Mathieu, Y.: Exploiting dual-output programmable blocks to balance secure dual-rail logics. Int. J. Reconfigurable Comput. **2010**, 375245 (2010). 12 pages. doi:10.1155/2010/375245
46. Schaumont, P., Tiri, K.: Masking and dual rail logic don't add up. In: CHES, Vienna, Austria, September 10–13. LNCS, vol. 4727, pp. 95–106. Springer, Berlin (2007)
47. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: CHES, Edinburgh, Scotland, UK, September. LNCS, vol. 3659, pp. 30–46. Springer, Berlin (2005)
48. Selmane, N., Bhasin, S., Guilley, S., Graba, T., Danger, J.-L.: WDDL is protected against setup time violation attacks. In: FDTC, Lausanne, Switzerland, September 6th, pp. 73–83. IEEE Comput. Soc., Los Alamitos (2009). doi:10.1109/FDTC.2009.40. In conjunction with CHES'09. Online version: http://hal.archives-ouvertes.fr/hal-00410135/en/
49. Shams, M., Ebergen, J.C., Elmasry, M.I.: Modeling and comparing CMOS implementations of the C-element. IEEE Trans. VLSI Syst. **6**(4), 563–567 (1998)
50. Shannon, C.E.: Communication theory of secrecy system. Bell Syst. Tech. J. **28**, 656–715 (1949)
51. Standaert, F.-X.: Secure and efficient implementation of symmetric encryption schemes using FPGAs, pp. 295–320 (2009)
52. Standaert, F.-X., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: EUROCRYPT, Cologne, Germany, April 26–30. LNCS, vol. 5479, pp. 443–461. Springer, Berlin (2009)
53. Standaert, F.-X., Örs, S.B., Preneel, B.: Power analysis of an FPGA: implementation of Rijndael: is pipelining a DPA countermeasure? In: CHES, Cambridge (Boston), MA, USA, August 11–13. LNCS, vol. 3156, pp. 30–44. Springer, Berlin (2004)
54. Standaert, F.-X., Rouvroy, G., Quisquater, J.-J.: FPGA implementations of the DES and triple-DES masked against power analysis attacks. In: Proceedings of FPL 2006, Madrid, Spain, August (2006)
55. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: another look on second-order DPA. Cryptology ePrint Archive, Report 2010/180. http://eprint.iacr.org/ (2010)
56. Suzuki, D., Saeki, M.: Security evaluation of DPA countermeasures using dual-rail pre-charge logic style. In: CHES. LNCS, vol. 4249, pp. 255–269. Springer, Berlin (2006)
57. Suzuki, D., Saeki, M.: An analysis of leakage factors for dual-rail pre-charge logic style. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **E91-A**(1), 184–192 (2008). doi:10.1093/ietfec/e91-a.1.184
58. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: DATE'04, Paris, France, February, pp. 246–251 (2004)
59. Tiri, K., Verbauwhede, I.: Place and route for secure standard cell design. In: Proceedings of WCC/CARDIS, Toulouse, France, August, pp. 143–158 (2004)
60. Tiri, K., Hwang, D., Hodjat, A., Lai, B.-C., Yang, S., Schaumont, P., Verbauwhede, I.: Prototype IC with WDDL and differential routing—DPA resistance assessment. In: Proceedings of CHES'05, Edinburgh, Scotland, UK, August 29–September 1. LNCS, vol. 3659, pp. 354–365. Springer, Berlin (2005)
61. Trichina, E., Seta, D.D., Germani, L.: Simplified adaptive multiplicative masking for AES. In: CHES, pp. 187–197 (2002)

62. Waddle, J., Wagner, D.: Towards efficient second-order power analysis. In: CHES. LNCS, vol. 3156, pp. 1–15. Springer, Berlin (2004)
63. Yu, P., Schaumont, P.: Secure FPGA circuits using controlled placement and routing. In: CODES+ISSS'07: Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, pp. 45–50. ACM, New York (2007). doi:10.1145/1289816.1289831

Chapter 5
True Random Number Generators in FPGAs

V. Fischer and F. Bernard

Abstract The issue of random number generation is crucial for the implementation of cryptographic systems in FPGAs. Random numbers are often used in key generation processes, authentication protocols, zeroknowledge protocols, padding, in many digital signature schemes, and even in some encryption algorithms. For these applications, security depends to a great extent on the quality of the source of randomness. The quality of the generated numbers is checked by statistical tests. In addition to the good statistical properties of the obtained numbers, the output of the generator used in cryptography must be unpredictable. For this reason, pseudo-random generators that are easily implementable in digital logic devices, including FPGAs, are not suitable for many cryptographic applications. In this chapter, we present the state-of-the-art of true random number generators in (reconfigurable) logic devices. We evaluate sources of randomness and the general principles used to extract and process randomness in FPGAs.

5.1 Introduction

Random number generators (RNGs) are one of basic cryptographic primitives used to design cryptographic protocols. Their applications include—but are not limited to—the generation of cryptographic keys, initialization vectors, challenges, nonces and padding values, and the implementation of countermeasures against side channel attacks. RNGs aimed at cryptographic applications must fulfill basic security requirements. First of all, their output values must have good statistical properties and be unpredictable. In modern designs, some additional features are required: the generator must be inherently secure, robust and resistant to attacks and/or tested on line using generator specific tests.

The security of cryptographic systems is mainly linked to the protection of confidential keys. In high end information security systems, when used in an uncontrolled environment, cryptographic keys should never be generated outside the system and

V. Fischer (✉)
Hubert Curien Laboratory—UMR CNRS 5516, Jean Monnet University, Saint-Etienne, France
e-mail: fischer@univ-st-etienne.fr

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_5, © Springer Science+Business Media B.V. 2011

they should never leave the system in clear. For the same reason, if the security system is implemented in a single chip (cryptographic system-on-chip), the keys should be generated inside the same chip. Implementation of random number generators in logic devices (including configurable logic devices) is therefore a major challenge.

There are three basic RNG classes used in cryptography:

Deterministic (pseudo-) random number generators (PRNG) are mostly fast and have good statistical properties. They are usually used as key generators in stream ciphers. Due to the existence of some underlying algorithms, PRNGs are easy to implement in logic devices. However, if the algorithm is known, the generator output is predictable. Even when the algorithm is not known but some of the generator output sequences have been recorded, its behavior during the recorded sequence can be used in future attacks. For this reason, pseudo-random number generators must be computationally secure (i.e., the underlying algorithm cannot be guessed computationally) and their seed value should never be reused. Binary sequences (counters) encrypted using a secure encryption algorithm are considered to be computationally secure. The reuse of the seed value can be avoided by saving the last counter value and using the following counter value next time.

Physical (true-) random number generators (TRNG) use physical processes to generate random numbers. If the underlying physical process cannot be controlled, the generator output is unpredictable and/or uncontrollable. The final speed of TRNGs is limited by the spectrum of the random signal and by the principle used to extract entropy from it (e.g. sampling frequency linked with the noise spectrum). The statistical characteristics of TRNGs are closely related to the quality of the entropy source, but also to the randomness extraction method. Because physical processes are subject to fluctuations, the statistical characteristics of TRNGs are usually worse than those of PRNGs.

Hybrid random number generators (HRNG) represent a combination of a (fast and good quality) deterministic RNG seeded repeatedly by a (slow but unpredictable) physical RNG. The designer has to find a satisfactory compromise between the speed of the generator and its predictability (by adjusting the time interval between seeds and the size of a seed).

In spite of the slower speed of TRNGs, they are more often used in cryptographic applications than PRNGs. It is interesting to note that TRNGs are the only cryptographic primitives that have not been subject to standardization up to now. However, before using the generator in practice, the principle and its implementation inside a cryptographic module has to be validated by an accredited institution as part of a security evaluation process. Generators that do not have a security certificate are considered to be insecure in terms of their use in cryptographic applications. For this reason, the study of main existing TRNG principles and of their characteristics presented in this chapter is of great interest.

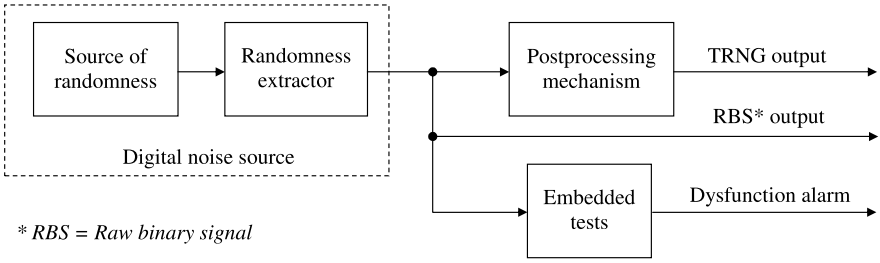


Fig. 5.1 TRNG general structure

5.2 Design of TRNGs

The general structure of a TRNG is depicted in Fig. 5.1. The generator should use an uncontrollable physical process as a source of randomness. Since physical phenomena used in TRNGs are mostly analog processes, some method enabling data conversion from analog to digital domain is usually necessary. An analog to digital conversion can be included in the randomness extraction procedure. The obtained unprocessed raw binary signal (so-called digital noise) can have low entropy and/or bad statistical properties. In this case, some post processing algorithms can be used to enhance the statistical parameters of the output bitstream. However, TRNG output post processing can sometimes mask a serious fault in the generator. Standard statistical tests may then fail to detect the masked weakness. Having the possibility to test the unprocessed digital noise is therefore recommended. The security of the generator can be increased if the statistical tests are applied on the fly and if they are designed to suit the generator's principle with particular reference to its potential weaknesses.

True random number generators use different sources of randomness and numerous principles to extract it. We next evaluate the main TRNG principles using three classes of characteristics: quality related parameters, security related parameters and design related parameters.

- *Quality-related parameters*
  - Source of randomness
  - Method of randomness extraction and entropy of the digital noise
  - Post processing method applied (optional)
  - Output bit rate and its stability
- *Security-related parameters*
  - Existence of a mathematical model
  - Inner testability
  - Security (robustness, resistance against attacks)
- *Design-related parameters*
  - Resource usage
  - Power consumption
  - Feasibility in logic devices and FPGAs
  - Design automation

It is important to note that these characteristics of TRNGs are not all equally important. Security parameters like robustness, availability of a stochastic model, testability, etc. always take priority in a data security system. Their weight in TRNG evaluation is much higher than that of other parameters like power consumption, bit rate, etc.

5.2.1 Sources of Randomness in Logic Devices

Logic devices are designed for the implementation of deterministic logic systems. Each unpredictable behavior in such a system (caused by a metastability, clock jitter, radiation errors, etc.) can have catastrophic consequences for the behavior of the overall system. Although unpredictable events due to the physical nature of the underlying technology are unavoidable, vendors of logic devices tend to minimize them. For this reason, the randomness extraction methods used in the TRNG design should be critically examined in order to keep up with the evolution of the underlying technology. Most logic devices do not contain any analog blocks, so the sources of randomness are related to the operation of logic gates. Several phenomena and their combinations can be used: variation in the delay of logic gates, analog behavior of logic gates between two logic levels (e.g. metastability), setup and hold time violation and thermal noise generated inside the device.

The instability of the delays of logic gates causes signal propagation variations over time. These variations can be seen as a clock period instability (the jitter) in clock generators containing delay elements assembled in a closed loop (ring oscillators). The variation in propagation time is also used in generators with delay elements in an open chain assembly. The chain is used to increase or adjust the total delay.

Since resistors and capacitors can be easily implemented in digital technology, the thermal noise generated in resistors can be used to modulate the frequency of a free running oscillator (RC oscillator). The thermal noise is thus converted to the time domain, where it can be easily extracted. However, this principle cannot be used in FPGAs, because appropriate structures are not available.

Some generators use the tracking jitter introduced by phase locked loops (PLLs) to generate random numbers. These so called analog PLLs are easy to implement in digital devices (including FPGAs), because the RC filter which is in such PLLs, is mostly the only "analog" block, can be easily implemented using the same technology. We can therefore consider a PLL-based TRNG as a generator that can be implemented in logic devices in general.

5.2.2 Randomness Extraction Methods

In logic devices that do not contain an analog block, randomness is often extracted by sampling a (clock) signal on the rising or falling edges of a reference (clock) signal using synchronous or asynchronous flip-flops (latches). The random bit stream

can be obtained in two ways: sampling random signals at regular intervals or sampling regular signals at random time intervals. In synchronous systems, the first method is preferable in order to guarantee a constant bit rate on the output.

The choice between synchronous and asynchronous flip-flops does not seem to be important in ASICs, but it is very important in FPGAs. This is because synchronous flip-flops are hardwired in logic cells as optimized blocks and their metastable behavior is consequently minimized. On the other hand, latches can usually only be implemented in Look up tables (LUTs) and are therefore subject to metastable behavior to a greater extent. Up to now, the behavior of synchronous and asynchronous flip-flops and their use for randomness extraction in FPGAs has not been sufficiently evaluated.

The randomness extraction method is usually linked to the basic principle of the generator and to the source of randomness. The randomness extraction procedure and postprocessing are sometimes merged in the same block and cannot be separated. In that case, the entropy of the randomness source is modified by postprocessing and cannot be measured or evaluated correctly.

In the true random number generator evaluation process, the source of randomness and the randomness extraction method are tightly linked and cannot be dissociated. For this reason, it is more reasonable to evaluate these two generator parameters together. The best way to do so is evaluating the entropy included in the digital noise.

5.2.3 Postprocessing the Raw Binary Signal

The evaluation of TRNGs is almost always based on statistical tests that are applied to random sequences produced by the TRNG. Sometimes the entropy source may have some weaknesses that lead to the production of non-random numbers (long sequences of zeros or ones). For this reason, postprocessing may be necessary to improve the statistical properties of random numbers, for example to increase entropy per bit, reduce bias and/or correlation.

The quality of the digital noise signal (the signal obtained in the randomness extraction block) can deteriorate for several reasons: (a) the entropy of the source is not high enough (this is often the case if metastability is used as a source of randomness); (b) the entropy, which is high in the original signal, is not well extracted; (c) the extracted samples are correlated. The entropy per bit at the output of the generator is mostly increased at the cost of reduction and/or variation in the bit rate.

XOR Corrector The XOR corrector is a simple linear function that applies an exclusive or operation on non-overlapping blocks of n bits in order to generate one output bit. It can dramatically reduce the bias on the generator output at the cost of reducing its bit rate n-times. However, the bias of the output bit stream is reduced only if the original bits are independent. The main advantages of the XOR corrector are its simplicity and the possibility to maintain a constant output bit rate. For a good analysis of the XOR corrector, the reader should refer to [12].

Von Neumann's Corrector Von Neumann's corrector is a simple non-linear function that takes successive pairs of bits and, if the bits are not the same, uses the first bit of the pair, and discards identical pairs. The output bit rate is therefore data dependent.

Although the input stream is stationary and may be biased, the output will be unbiased. However, if the original stream is autocorrelated, the output may still be autocorrelated. It should also be noted that von Neumann's corrector will produce a biased output if the input stream features a cycle with period 2. If the corrector is implemented in hardware, it may interfere with the generator and result in exactly this type of occurrence.

Linear Feedback Shift Registers (LFSRs) LFSRs are used in many random bit stream generators. There are several reasons for this: (1) LFSRs are easy to implement in hardware; (2) they can produce sequences of large period; (3) they can produce sequences with good statistical properties; and (4) because of their structure, they can be easily analyzed using algebraic techniques.

A LFSR of length L consists of L delay elements each capable of storing one bit and having one input and one output and a clock that controls the movement of data. During each unit of time, the following operations are performed: (i) the content of the first delay element is output and forms part of the output sequence; (ii) the content of element i is moved to stage $i-1$ for each i, $1 \leq i \leq L-1$; and (iii) the new content of the last delay element is the feedback bit that is calculated by summing modulo 2 the previous contents of a fixed subset of elements (depending on the underlying polynomial) [32].

Resilient Functions Resilient functions are specific functions used in cryptography and coding theory. They derive from Boolean functions. The study of Boolean functions is very important in cryptography (particularly in the design of symmetric key algorithms). For more details on Boolean functions in general (degree, Normal Algebraic Form, Möbius transform, Walsh-Hadamard transform, etc.), the reader should refer to [21].

In more informal terms, resilient functions are suitable for postprocessing because the knowledge of any m values of the input to the function does not allow anyone to make a better than random guess at the output [44]. Resilient functions are based on the same principle as error correcting codes. While error correcting codes are used to suppress random errors, resilient functions are used to extract these random bits. The raw binary signal (digital noise) can be considered as a bit stream containing a redundancy and the resilient function reduces the bit rate while extracting random bits. In this way, the resilient function increases the entropy per bit at the generator output. For more details on linear codes and resilient functions in error correcting codes theory, the reader should refer to [9] and [24].

Encryption of the Digital Noise Signal This kind of the digital noise postprocessing uses both diffusion and confusion properties of cryptographic functions. The perfect statistical characteristics of most of the encryption algorithms can be

used to mask generator imperfections. One of advantages of this approach is that the encryption key can be used as a cryptographic variable to dynamically modify the behavior of the generator. Although this kind of postprocessing block (the cipher) is rather complex and expensive, the TRNG can reuse (share) the cipher that is used for data encryption.

Hashing of the Digital Noise Signal One of the most time consuming but also one of the most secure methods is cryptographic postprocessing based on hash functions such as MD5, SHA-1 or others. It uses diffusion and one-wayness (as opposed to encryption of the raw binary signal) properties of hash functions to ensure the unpredictability of bits generated by the TRNG if a total breakdown of the noise source occurs. In this case, due to the non-linearity property of hash functions, the TRNG will behave like a PRNG.

5.2.4 Output Bit Rate

The speed is a secondary parameter (after security) in many cryptographic applications. Output bit rates from hundred kilobits per second up to 1 megabit per second are usually sufficient. However, there are some speed critical data security applications for which high speed generators are required. For example, Quantum cryptography requires a high bit rate (up to 100 megabits per second) because of the very low efficiency of key data transmission over the low-power optical channel.

High speed telecommunication servers are a second example. They need to generate session keys on a regular high speed basis (tens of megabits per second). For example, a 10-Gbit Ethernet hub/server would need about 20 Mbits/s random bits to generate one 128-bit session key for each 64 kB data block in order to be able to resist side channel attacks.

High speed telecommunication servers can be given as a second example. They need to generate session keys on a regular high speed basis (tens of megabits per second). For example a 10-Gbit Ethernet hub/server would need about 20 Mbits/s random bits to generate one 128-bit session key for each 64 kB data block in order to be able to face side channel attacks.

Another aspect of the output bit rate that has to be considered is its variability. Some generators give random numbers periodically, others generate output in irregular time intervals. In the second case, a FIFO is required to accumulate the generated numbers. Another solution is to estimate the smallest bit rate available at the output and to sample the output at this rate. The disadvantage of the first solution is that, depending on the mean output bit rate and the need for random numbers, the FIFOs sometimes need to be very big. The disadvantage of the second solution is that if the estimated bit rate is incorrect, the random numbers may not be sometimes available at the output.

5.2.5 Modeling TRNGs

The main part of a security certification process deals with evaluation of the design and implementation of the generator. The aim of the evaluation is to quantify the entropy per random bit [34]. However, entropy is a property of random variables and not of observed realizations (random numbers). In order to quantify entropy, we thus need to analyze the distribution of the random variables, e.g. by the use of a stochastic model. The stochastic model specifies a family of probability distributions of random variables that enables verification of a lower entropy bound for the raw binary signal. In [34], the authors use a model that gives a lower bound of the average conditional entropy per internal random number. The value given by the model can be used to test the entropy of the generated numbers in real time.

5.2.6 Testability

Inner testability means that the generator structure enables evaluation of the entropy of the raw binary signal (the raw binary signal must be available). This functionality is required in recent TRNG evaluation procedures [45]. However, when randomness extraction and post processing are merged in the same process, the unprocessed random signal is not available. Even when this signal is available, it is sometimes composed of a pseudo random pattern combined with a truly random bit stream. The pseudo random pattern makes statistical evaluation of the signal more difficult. For this reason, we propose a new testability level: an absolute inner testability. The output of the generator featuring absolute inner testability does not include a pseudo random pattern and contains only a true random bit stream. The functional simulation output of the generator with absolute inner testability is always zero. If (for some reason) the source of randomness fails in a hardware generator, the test output will be zero. This fact can be used to secure the generator.

5.2.7 Security Evaluation

It is often very difficult (and sometimes impossible) to build a stochastic model for a particular generator. In that case, another approach can be used to validate the use of the generator in cryptographic applications. This approach is based on the analysis of the impact of the changing environment or an attack on the generator (security evaluation). There are three possibilities: (i) proof exists that the generator cannot malfunction as the result of any attack or of a changing environment, (ii) neither security proof nor attack exists, (iii) some attack on a particular generator has been reported.

5.2.8 Resource Usage

To evaluate various TRNG principles, it is important to analyze the resources needed for generator hardware implementation. Generally speaking, all kinds of resources available in FPGAs can be used to generate random numbers: LUT based or multiplexer based logic cells, embedded memory blocks, clock blocks with PLLs and DLLs, embedded RC oscillators, hardwired multipliers, programmable interconnections, etc.

FPGAs have many logic cells, so the use of logic cells (the logic area) is usually not a problem. The topology and electrical parameters of programmable interconnections are strongly technology dependent. Many TRNG designs require the designer's manual intervention during placement and routing (P/R). Concerning P/R, some designs can be easily implemented in one FPGA family, but are difficult or impossible to implement in others. The choice and the number of embedded hardwired blocs is usually much more limited (PLLs, RC oscillators, multipliers, memory blocks) and varies with the vendor and the technology. The use of hardwired blocks can thus be a limiting factor for reusability of the TRNG principle.

5.2.9 Power Consumption

The power consumption of the generator is linked to its randomness source (e.g. the oscillator), to the clock frequency used and to algorithm agility. In power critical applications, the generator can be stopped when not in use. However, the possibility to control bit stream generation can be used to attack the generator.

5.2.10 Feasibility in FPGAs

Compared to the implementation of TRNGs in ASICs, their implementation in FPGAs is much more restricted. Many TRNGs implemented in ASICs use analog components to generate randomness (e.g. chaos based TRNGs using analog to digital converters, free running oscillator based generators using thermal noise from diodes and resistors, etc.) and to process randomness (e.g. operational amplifiers, comparators, etc.). Most of these functional blocks are usually not available in FPGAs, although some of them may be available in selected families, e.g. RC oscillators in Actel Fusion FPGA, analog PLLs in most Altera and Actel families but not in old Xilinx families. From the point of view of their feasibility in FPGAs, some generators are not feasible or are difficult to implement in FPGAs, some are feasible in selected FPGAs and the most general principles are feasible in all FPGAs.

5.2.11 Design Automation

The fact that the generator uses resources that are available in FPGAs does not automatically mean that it can be implemented in this kind of device. The range of tolerance of some technology parameters can be such that it prevents reliable implementation of the generator. The parameter that most often limits generator implementation in FPGAs is the availability of routing resources and their characteristics. Some generators require perfectly balanced routing. This necessitates perfect control of the module placement (e.g. symmetrical placement of two modules in relation to another module) and routing. While most FPGA design tools allow precise control of placement, the routing process is difficult or impossible to control (e.g. in the Actel family). Even when routing can be partially or totally controlled (e.g. Altera and Xilinx families), the delays in the configurable routing net vary so much from device to device that it is impossible to balance module interconnections in a general manner and the design will be device dependent, i.e. it has to be balanced manually for each device. Such manual intervention is not acceptable from the point of view of the practical implementation of the generator. The best generators (very rare) can be mapped automatically (without manual intervention) in all FPGA families. From practical point of view, it is still acceptable if the implementation of the generator requires manual intervention (P/R) for each family and/or type of device. However, generators that require manual optimization on a per device basis are not acceptable for industrial applications.

5.3 Quality and Security Issues in TRNG Design

The output of a good TRNG should be indistinguishable from the output of an ideal TRNG, independently of operating conditions and time. The quality of the generator output bit stream and its security parameters including robustness against aging, environmental changes, attacks, existence of selftest and online tests are very important in the TRNG design.

5.3.1 Operating Conditions and Quality of the TRNG Output

Good TRNGs are supposed to give a uniformly distributed stream of 0s and 1s at their output. The quality of the generator output is tightly linked with the quality of the source of randomness and to the randomness extraction method used. The spectral characteristics of the source of randomness and the randomness extraction determine the principal parameters of the generated bit stream: the bias of the output bit stream, correlation between subsequent bits, visible patterns, etc. While some of these faults can be corrected by efficient post processing, it is better if the generator inherently produces a good quality raw bit stream.

If the extractor samples the source of randomness too fast, adjacent bits could be correlated. For this reason, it is good practice to check the generated bit stream for a short term auto correlation. It is also possible that the digital noise exhibits some other short term dependencies, which need to be detected by some generator specific tests.

The behavior of the generator is often influenced by external and/or internal electrical interferences. The most obvious effect of this will be discrete frequencies from the power supply and from various internal signals appearing in the noise spectrum.

The spectrum of the generated noise signal can be significantly influenced by a low frequency 1/f noise caused by semiconductors. Furthermore, the high frequencies from the noise spectrum may be unintentionally filtered out by some internal capacities.

Some generators can feature so called bad spots. Bad spots are short periods when the generator ceases to work, possibly due to electrical interference or to extreme excursions of the overloaded part of the generator circuitry.

Another dangerous feature of the generator can be a back door, which refers to the deviations from uniform randomness deliberately introduced by the manufacturer. For example, let us suppose that instead of using some physical process, the generator would generate a high quality pseudo random sequence with a 40-bit seed. It would be impossible to detect this behavior by applying standard statistical tests on the output bit stream, but it would be computationally feasible for someone who knows the back door to guess successive keys.

5.3.2 TRNG Evaluation and Testing

TRNG must undergo the quality and security evaluation process before it can be used in data security systems. For a long time, TRNG testing was limited to evaluating the ability of generated sequences to pass a battery of statistical tests (FIPS 140-1 [17], NIST SP 800-22 [43] or DIEHARD [40, 41]).

5.3.2.1 General Strategy of Statistical Testing

The role of TRNG statistical testing is to evaluate the hypothesis that the generator is indeed random. This hypothesis is called the null hypothesis (H_0), because it is based on our beliefs, in the absence of evidence. The statistical tests should provide the evidence that the hypothesis is true. If the tests are not successful, the hypothesis is rejected, otherwise it is not (but it shouldn't be completely accepted because there is no evidence that the hypothesis is true). For each existing test, a relevant randomness statistic is chosen and used to determine the rejection of the null hypothesis. Under the assumption of randomness, such a statistic has a distribution of possible values. A theoretical reference distribution of this statistic under the null hypothesis is determined by mathematical methods. From this reference distribution, a critical

value is determined. A statistical value is then computed on the sequence and is compared to the critical value. If the statistical value exceeds the critical value, the null hypothesis for randomness is rejected.

A good true random number generator may produce a sequence that does not appear to be random while a bad TRNG may produce an apparently random sequence. Two different kinds of errors are usually considered:

- Type 1 error: when the statistical test rejects a sequence that is in fact random (false reject).
- Type 2 error: when the statistical test accepts a sequence that isn't in fact random (false acceptance).

The probability of a Type 1 error is denoted α and the probability of the Type 2 error is denoted β. Because of these errors, no statistical test can specify with certainty whether a sequence is random or not. The significance level of the test is defined by the probability of the Type 1 error (α).

For each statistical test, a P-*value* is computed. This P-*value* is the probability that a perfect (ideal) random number generator would have produced a sequence seemingly less random than the sequence that was tested. If the P-*value* is greater than α, then the null hypothesis is accepted. Otherwise, it is rejected.

For cryptographic applications, α is usually chosen in interval $[0.001; 0.01]$. An α of 0.001 indicates that one sequence in 1000 sequences is expected to be rejected by the test if the sequence was random. For a $P\text{-}value \geq 0.001$, a sequence would be considered to be random with 99.9% confidence, whereas for a $P\text{-}value < 0.001$, a sequence would be considered to be non-random with 99.9% confidence.

Even though statistical tests are required to evaluate the quality of the generated sequence, they are not able to distinguish between pseudo random data generated by a deterministic generator and truly random data from a physical TRNG. For this reason, a new evaluation methodology for physical random number generators was proposed by the BSI (Bundesamt für Sicherheit in der Informationstechnik) in 2001—AIS31 [33, 45].

5.3.2.2 AIS31: Methodology for TRNG Evaluation

AIS31 is an evaluation methodology for true (i.e. physical) random number generators. In contrast to the standard methods that tests only TRNG output, AIS31 also tests the raw binary signal. This new approach is motivated by the fact that the post processing can mask serious defects of the generator. If a stochastic model of the physical randomness source is available, it can be used in combination with the raw signal to estimate entropy and bias depending on random input variables and on the generator principle.

The raw binary signal is also used in *Online tests*. Online tests should be applied to the digital noise signal while the generator is running. They provide ways to stop the TRNG (at least temporarily) when a conspicuous statistical feature is detected. A special kind of online test required by the AIS31 is a "total failure test" or *Tot test* that should be able to immediately detect total failure of the generator.

AIS31 introduces two TRNG functionality classes depending on the security requirements: $P1$ and $P2$. Class $P1$ applications include:

- challenge response protocols,
- transmission of initialization vector in clear,
- seeds for PRNGs class $K1$ and $K2$ (AIS 20).

Class $P2$ contains:

- symmetric and asymmetric cryptographic keys,
- padding bits,
- zero knowledge proofs,
- seeds for PRNGs class $K3$ and $K4$ (AIS 20).

The functionality class $P2$ requirements are more restrictive. For example, in $P2$, the generated random numbers are practically impossible to determine even if the predecessors or successors are known.

Nine statistical tests are proposed in AIS31 for use at different stages of the TRNG evaluation. Five of them (tests *T0* to *T4*) test generated numbers (TRNG output):

- *T0*: Disjointness test—tests the coincidence of non-overlapping patterns in a sequence,
- *T1*: Monobit test—tests the bias of a bit stream,
- *T2*: Poker test—tests the occurrence of non-overlapping 4-bit blocks in the bitstream,
- *T3*: Runs test—tests the number of 0-runs (or 1-runs) of length $i < 6$ and $i \geq 6$,
- *T4*: Long runs test—searches for runs of length $i \geq 34$, while no long runs are allowed.

The last four tests (tests *T5* to *T8*) test the raw binary signal, while tolerating some weaknesses. The rejection limits in these tests are set according to the knowledge that a considerable bias could be present in the bit stream.

- *T5*: Autocorrelation test,
- *T6*: Uniform distribution test,
- *T7*: Comparative test for multinomial distribution,
- *T8*: Coron's entropy test.

A TRNG is said to be *P1-compliant* if it satisfies class P1 requirements:

- Generated random vectors must pass the disjointness test *T0*.
- Output random bit streams have to pass selected statistical tests, e.g. *T1* to *T4*.
- The raw binary signal should be tested for the total failure of the source of randomness (fast total failure test).
- When intended for high strength security applications, the statistical properties have to be verified in different operating conditions (temperature, voltage).
- Online test(s) must check the internal random numbers on demand or at regular intervals.

A TRNG is said to be *P2-compliant* if it satisfies $P1$ requirements plus following class $P2$ requirements:

- The bias of the digitized noise (raw signal) should be ≤ 0.025.
- Tests $T5$ to $T8$ applied on the raw binary signal have to pass.
- Post processing (if used) must not reduce the average entropy per bit.
- A set of special statistical tests has to be applied on each TRNG start (startup test).
- For high strength mechanisms, the statistical parameters and in particular the entropy of the digitized noise signal must be tested in different operating conditions.
- For high strength mechanisms, the TRNG must itself trigger an online test at regular intervals.

Evaluating TRNGs is a difficult task. Clearly it should not be limited to testing the TRNG output. Following the AIS31 methodology, the designer should also provide a stochastic model based on the noise source and the extraction process and propose statistical and online tests suited to the generator's principle. AIS31 gives a one proposed methodology. It does not favor or exclude any reasonable TRNG design. The applicant can also substitute alternative evaluation criteria, however these must be clearly justified.

5.4 Jitter as a Source of Randomness

As mentioned in the previous section, clock jitter is a source of randomness that can be reliably used in logic devices. Jitter is a general term used to specify clock or oscillation uncertainty in the time domain. Jitter can be defined as [48] a short-term variation of an event from its ideal position. In general, it is the variation in time of the zero crossing (rising or falling edge) of the clock signal. This definition helps understand what jitter is, but the situation is more fuzzy when we need to quantify the jitter. The problem is that different measurement techniques are used in different applications and that several authors define them differently according to their specific needs.

5.4.1 Jitter Quantification

To avoid confusion, let us first define clock jitter and its common measurements over time intervals. The instantaneous output voltage of an oscillator can be expressed [58] in the frequency or time domain as

$$V(t) = V_0 \sin(\omega_0 t + \varphi(t)) = V_0 \sin\left(\frac{2\pi}{T_0}(t + \delta(t))\right), \tag{5.1}$$

where $\varphi(t)$ is the phase deviation in the frequency domain expressed in radians and $\delta(t) = \varphi(t)/\omega_0$ is the time deviation (the jitter) expressed in seconds. For clock

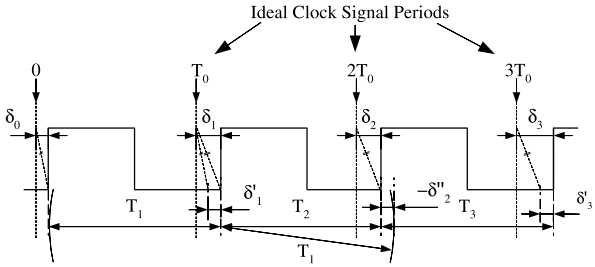


Fig. 5.2 Clock jitter measurements: Phase jitter (δ_n), Period jitter (δ'_n) and Cycle-to-cycle jitter (δ''_n)

applications, time domain measurements are preferable, since most specifications of concern involve time domain values. There are three basic measurements of the jitter in the time domain: phase jitter, period jitter, and cycle-to-cycle jitter.

Phase jitter is defined as the phase advance of the observed clock from an ideal clock with period. It is a phase measurement made at discrete time intervals nT_0, $n = 0, 1, 2, \ldots, \infty$. An ideal clock signal would have a cycle occurrence (e.g. rising edge) when $t = nT_0$, but the noisy clock signal with the time deviation $\delta(nT_0)$ will have a time shift error. We can express the cycle occurrences of the noisy clock using the ideal period

$$t_n = nT_0 + \delta(nT_0) = nT_0 + \delta_n. \tag{5.2}$$

Phase jitter δ_n is defined as the difference between noisy clock period and ideal period time nT_0. It is thus equivalent to a discrete measurement of the time deviation sampled at each ideal clock period:

$$\delta_n = t_n - nT_0. \tag{5.3}$$

Period jitter δ'_n is defined as the difference between measured adjacent clock periods and the ideal clock period T_0. It can be considered the first difference function of the phase jitter. Like the phase jitter, it is a discrete measurement, made at period intervals and is defined as

$$\delta'_n = (t_n - t_{n-1}) - T_0 = \delta_n - \delta_{n-1}. \tag{5.4}$$

Cycle-to-cycle jitter is the measured difference between two successive clock periods

$$\delta''_n = (t_n - t_{n-1}) - (t_{n-1} - t_{n-2}) = \delta'_n - \delta'_{n-1}. \tag{5.5}$$

It contains information about the short term dynamics of the jitter evolution and can be obtained by applying the first order difference to the period jitter or the second order difference to the phase jitter. The relationship between the phase jitter, the period jitter and the cycle-to-cycle jitter is depicted in Fig. 5.2.

5.4.2 Jitter Behavior over Time

Phase noise measures an oscillator's uncertainty in the frequency domain, while jitter measures this uncertainty in the time domain. Now we analyze in more detail how jitter behaves over time.

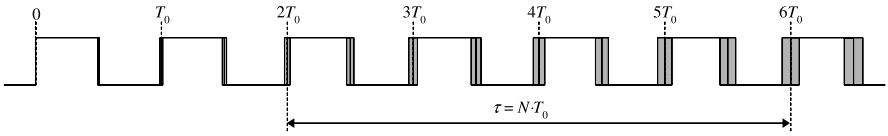


Fig. 5.3 TIE as a difference between two clock edges separated by an ideal edge delay of N clock periods. Its mean square average value is known as timing jitter

Consider the time interval between two arbitrarily chosen rising clock edges separated by N clock periods. Let the ideal rising edge delay for the N clock cycles be written as $\tau = N \cdot T_0$. The two actual clock edges and their time separation are now influenced by the fluctuating time deviation $\delta(t)$ and can be written as:

$$\begin{aligned} t_{ris1} &= t + \delta(t), \\ t_{ris2} &= (t + \tau) + \delta(t + \tau), \\ t_{ris2} - t_{ris1} &= \tau + (\delta(t + \tau) - \delta(t)). \end{aligned} \tag{5.6}$$

The accumulated error between the two clock edges is known as the time interval error (TIE). The TIE for an N clock period interval is defined as

$$TIE(t, \tau) = \delta(t + \tau) - \delta(t) = \frac{1}{\omega_0}(\varphi(t + \tau) - \varphi(t)). \tag{5.7}$$

In other words, TIE is a measure of the accumulated timing error between endpoints t and $t + \tau$ over the $\tau = N \cdot T_0$ time interval (see Fig. 5.3). If the measurement is initialized such that $t = 0$ and $\delta(t) = 0$, the TIE is identical to the phase jitter $\delta_n = \delta(N \cdot T_0)$. TIE and phase jitter are thus sometimes used interchangeably. Taking the mean square average of the TIE, which is called the timing jitter, gives

$$\overline{TIE(t, \tau)^2} = \frac{1}{\omega_0^2}[\overline{\varphi(t + \tau)^2} - 2\overline{\varphi(t + \tau)\varphi(t)} + \overline{\varphi(t)^2}]. \tag{5.8}$$

It shows the cumulative effect of the jitter over time if one assumes knowledge of ideal clock locations. The TIE can be obtained by integrating the period jitter after subtracting the ideal period from each measured period. This metric diverges over time.

5.4.3 Jitter Composition

Clock jitter usually has two components (see Fig. 5.4): *random jitter*, which is caused by a non-deterministic phenomenon like thermal/flicker noise and *deterministic jitter*, which is caused by some deterministic processes.

Random jitter obeys the central limit theorem and it has a Gaussian probability distribution function (PDF). As its name suggests, it has a random behavior and statistical tools (e.g. means, standard deviation) are thus often used to measure it. Random jitter is caused by the sum of many independent contributors inherent to

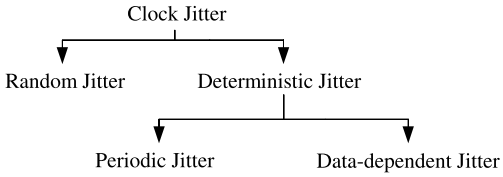


Fig. 5.4 Clock jitter composition

any electric circuit: thermal vibrations of semiconductor crystal structures, thermal vibrations of conductor atoms and many other minor contributors [37, 49, 50, 57].

Deterministic jitter has two other components (see Fig. 5.4): periodic jitter and data dependent jitter. Deterministic jitter is typically caused by power supply variations, cross talks, electromagnetic interference (EMI), simultaneous switching outputs and other regularly occurring interference signals.

The main confusion when one is confronted with jitter measurement and quantification is that some jitter metrics apply only to random jitter and not to deterministic jitter, while the latter is present almost all the time.

5.4.4 Jitter Measurement

5.4.4.1 External Methods of Jitter Measurement

A lot of measuring equipment enables precise jitter measurement outside the device. The jitter is mostly represented by a histogram, from which one can see if it has some deterministic components or not. If it does not, the jitter can be quantified from the histogram using the standard deviation σ, which represents 68.28 percent of the samples in the histogram. The jitter size obtained from σ is called RMS jitter. This kind of jitter evaluation is valid only for Gaussian jitter distributions (see Fig. 5.5). If the jitter histogram differs from an ideal Gaussian distribution, this means that if

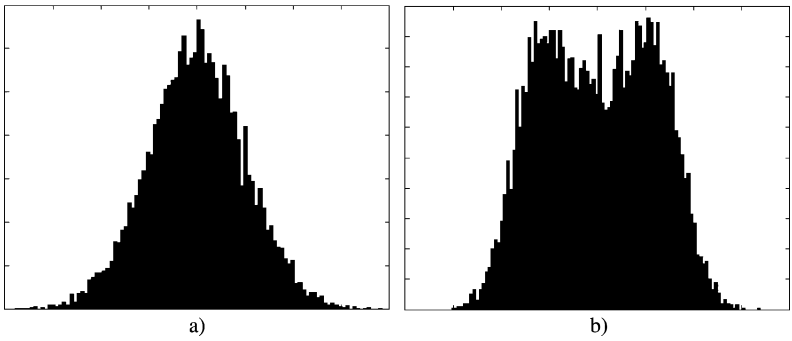


Fig. 5.5 Jitter histograms: jitter with random component only (**a**), with both a random and a deterministic component (**b**)

the jitter contains some deterministic components, measuring the RMS jitter from the histogram is no longer possible.

Typically, we can assume that the random (stationary) noise sources will result in average phase deviation values that are independent of the time shift. They therefore share the same RMS value $\varphi_{rms}^2 = \overline{\varphi(t)^2}$, and we can consider the TIE from Eq. (5.8) to be a function of the observation interval only, i.e. $\overline{TIE(t,\tau)^2} = \overline{TIE(\tau)^2}$. The remaining term from Eq. (5.8) represents the autocorrelation function of the phase deviation $R_\varphi(\tau) = \overline{\varphi(t+\tau)\varphi(t)}$. If the jitter contains only the random component, the mean-square average value for TIE equates to a σ^2 variance of the Gaussian distribution of the jitter. In this way, the timing jitter (or accumulated jitter) can be represented by the variance of the TIE, $\overline{TIE(\tau)^2}$, which can be written as

$$\sigma_{TIE}^2(\tau) = \frac{2}{\omega_0^2}[\varphi_{rms}^2 - R_\varphi(\tau)]. \tag{5.9}$$

The N-cycle timing jitter is therefore a function of RMS phase jitter, which does not depend on the time interval, and of the interval dependent autocorrelation.

It can be seen that the period jitter is a timing jitter over an interval of one period ($\tau = T_0$), i.e.

$$\sigma_{PER}^2(\tau) = \sigma_{TIE}^2(T_0). \tag{5.10}$$

The timing jitter measured over one signal period is commonly used as a fundamental jitter measurement for free running oscillators with an unbounded RMS phase jitter.

The cycle-to-cycle jitter represents the error between adjacent periods. It is equal to the difference between the TIE at time $\tau = 2T_0$ and $\tau = T_0$. The variance of the cycle-to-cycle jitter can be calculated as:

$$\sigma_{CTC}^2(\tau) = 4\sigma_{PER}^2 - \sigma_{TIE}^2(2T_0). \tag{5.11}$$

Because the histogram representation does not provide any information about the time evolution of the jitter, it is sometimes useful to represent the time dependence of the jitter, i.e. the period jitter (or cycle-to-cycle jitter) vs. time. When it is necessary to quantify the jitter that is composed of random and deterministic components, we usually measure the peak-to-peak jitter, which is the difference between the biggest and the smallest value in the jitter population over a certain time period τ. Due to the unbounded nature of the random jitter, the peak-to-peak value increases with τ. We can also try to separate or extract the random jitter from the composed jitter, but in this case, precise modeling of the entire system is necessary to avoid incorrect separation.

In [27, 28] Hajimiri and Lee give a complete theory of modeling the phase noise in oscillators and how to relate it to the timing jitter. Although this work refers to low level electrical modeling, it can serve as a starting point for high level modeling.

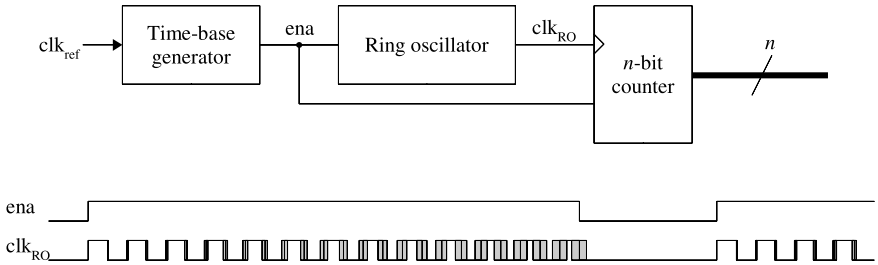


Fig. 5.6 Timing jitter measurement embeddable in logic devices

5.4.4.2 Embedded Methods of Jitter Measurement

The clock jitter measured outside the FPGA is significantly influenced by the device input/output circuitry. It is therefore useful to measure the jitter inside the device. The method published in [53] is presented in Fig. 5.6.

The time base generator generates a measurement interval (signal ENA) enabling ring oscillator output. The clock generated in the ring oscillator is used to increment an n-bit counter, which is stopped at the end of measurement interval. The output value of the counter is proportional to the ring oscillator frequency. As shown in Sect. 5.4.2, the TIE can be derived from the obtained counter value.

5.5 Random Number Generators in Logic Devices

In this section, we discuss in detail the main TRNG designs reported in the literature. Most of them use digital technology and can be (or have already been) implemented in FPGAs. However, many other design that exploit analog components or other technologies that cannot be directly implemented in FPGAs are not included in our analysis. For example, the designs proposed in [1, 32, 34, 42, 56, 59] use noise sources, other designs e.g. [2, 4, 5] use precise delay elements and/or analog circuitry that are not feasible or not available in reconfigurable logic devices.

We group TRNG designs implementable in FPGAs according to the noise source exploited or according to the method used to extract randomness. According to these criteria, we can distinguish seven groups of TRNG designs:

- Generators that sample independent jittery clock(s),
- Stateless generators using gated ring oscillators,
- Generators using coherent sampling,
- Generators that combine pseudo and true randomness,
- Generators using open delay chain,
- Generators based on metastability,
- Other generators.

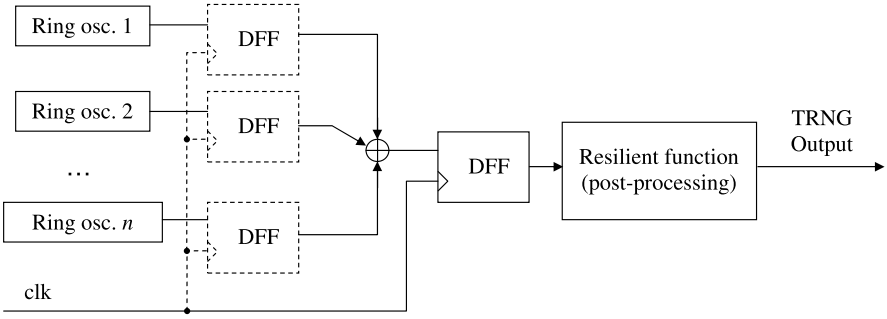


Fig. 5.7 Sunar *et al.*'s TRNG and modifications proposed by Wold and Tan (*dashed lines*)

5.5.1 Generators that Sample Independent Jittery Clock(s)

In this group of TRNGs, the output of one or more high frequency oscillators (mostly ring oscillators) is sampled using synchronous or asynchronous *D* flip-flops [3, 36, 52]. The sampling frequency is mostly a reference clock signal (e.g. from a quartz oscillator).

Sunar *et al.*'s Generator The principle of the random number generator proposed by Sunar *et al.* [47, 60] is presented in Fig. 5.7. This TRNG uses *n* ring oscillators (114 for selected Xilinx FPGA), and each oscillator comprises 13 inverters. The number of oscillators is selected according to the size of the measured jitter and the sampling frequency. The outputs of oscillators are XOR-ed together in order to obtain a high frequency raw signal. This signal is sampled using a low frequency reference clock in a *D* flip-flop (DFF) to obtain a digitized random signal. Sunar *et al.* propose a mathematical proof that defines the relationship between the number of oscillators, sampling frequency and the digital noise entropy. Based on this estimated entropy, the digital noise signal is postprocessed using a resilient function based on a cyclic code [256, 16, 113].

The work published in [44] was the first practical application of the principle proposed by Sunar *et al.*, but the number of inverters and the number of ring oscillators were not the same (three inverters, 210 oscillators in a robust version and 110 oscillators in an economic version). The resilient function remained unchanged.

The main advantage of this TRNG principle is that it does not depend on the technology and can be easily implemented in all FPGA families. It does not require any manual intervention during the mapping procedure. It uses an acceptable amount of resources and has a relatively high output bit rate. Thanks to the use of a resilient function, the output bit rate is constant. Since many oscillators used in the design oscillate continuously, the power consumption of the generator is relatively high.

Although the authors of the original generator claim that it is the only existing generator that is provably secure, it appears that their mathematical proof is based on unrealistic physical assumptions (a single XOR gate cannot linearly process a huge number of high speed signals, and ring oscillators are not totally independent as assumed in the proof) [14].

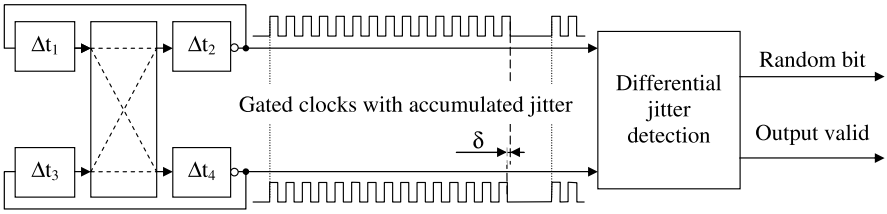


Fig. 5.8 TRNG of Bucci *et al.*

In [20], Wold and Tan proposed a significant modification of the generator—they showed that adding a flip-flop to the output of each ring oscillator (as it is depicted in Fig. 5.7) solves the problem of the speed of the XOR gate. However, their suggestion on how to reduce the number of rings is security critical and should consequently be avoided.

The generator is inner testable, but not absolutely inner testable: ring oscillators create a pseudo random pattern that cannot be removed. It is therefore possible that the manipulations or attacks on the generator would not be detected.

In [20], Fischer *et al.* showed that the generator can become a subject of attack because the local Gaussian and the global deterministic jitter accumulate differently. Since the sampling clock comes from a fixed external reference, the pattern on the XOR gate output strongly depends on the deterministic jitter. Fortunately, the security of the generator can be improved by using a reference signal that comes from another ring oscillator implemented in the same device rather than a fixed external reference clock signal. According to these authors, the improvement in security is due to the fact that the deterministic jitter (which represents the means of attack) will accumulate in the source ring oscillators and in the reference ring oscillator in the same way. In this way, the accumulated deterministic jitters compensate for each other.

5.5.2 Stateless Generators Using Ring Oscillators

The class of stateless generators is defined in [3]. The aim of the design of stateless random number generators is to remove the dependence of the next state of the generator on its previous state (i.e. to remove predictability). The internal state of ring oscillator based TRNGs described in the previous section depends on the evolution of mutual phases of ring oscillators. This kind of generator contains some memory elements, the set of current phases of all oscillators. In order to remove dependence on the previous state, all the rings using gated ring oscillators can be regularly "restarted". The stateless TRNGs obtained are mostly absolutely inner testable.

Bucci *et al.*'s Generator The principle of random number generator proposed by Bucci *et al.* [3, 6] is presented in Fig. 5.8. The generator exploits relative jitter between two gated ring oscillators sharing the same delay elements (Δt_1 to Δt_4).

These four delay elements are connected by means of a symmetrical switching matrix in order to obtain two ring oscillators sharing the same delay elements and thus featuring an identical mean period. The cross coupled oscillators start to oscillate synchronously and their relative accumulated jitter increases as long as they continue running. A detector enables detection of the differential jitter above a given threshold (an inverter propagation delay). Depending on the sign of the detected jitter, a random bit is generated and both oscillators are stopped and restarted to begin a new cycle. With this approach, the accumulated relative jitter depends on the noise generated inside the circuit and so the data rate of the generator is not constant.

This generator was used in a 0.13 μm CMOS Standard Cells technology from Infineon Technologies. The authors claim that after about 20 complete periods, the relative jitter is sufficient for the generation of a new bit. The generator requires post processing. Since it uses a von Neumann corrector, the output rate is irregular. It requires very few hardware resources. The gated ring oscillators prevent randomness generation and power consumption is thus reduced. Because the generator is reset at the beginning of each cycle, it is stateless and so absolutely inner testable. Although the authors does not propose a model, it seems that, given the relative simplicity of the generator, a stochastic model could be developed. Following the principle of differential jitter accumulation, we presume the generator would be robust against attacks.

Because the generator uses only logic resources, it should be (at least theoretically) implementable in FPGAs. However, it requires manual intervention to implement perfectly symmetrical cross coupled oscillators. This symmetry is very difficult or even impossible to obtain in certain FPGA families.

5.5.3 Generators Using Coherent Sampling

The class of generators based on coherent sampling uses two or more clock generators with related output frequencies. Depending on the frequency ratio (or the phase difference) and the method of randomness extraction, these generators can be absolutely inner testable. The principle is simple and can be modeled relatively easily.

Fischer and Drutarovsky's Generator The basic principle behind the method proposed by Fischer and Drutarovsky in [18] is to extract randomness from the jitter of the clock signal synthesized in an embedded analog phase locked loop (PLL).

The jitter is detected by sampling a reference (clock) signal using a rationally related (clock) signal synthesized in the PLL. In order to extract randomness, the reference signal has to be sampled near the edges influenced by the jitter. The basic structure of the proposed generator is depicted in Fig. 5.9.

Let *CLJ* be an on chip PLL synthesized rectangular clock waveform with the frequency

$$F_{CLJ} = F_{CLK} \frac{K_M}{K_D}, \tag{5.12}$$

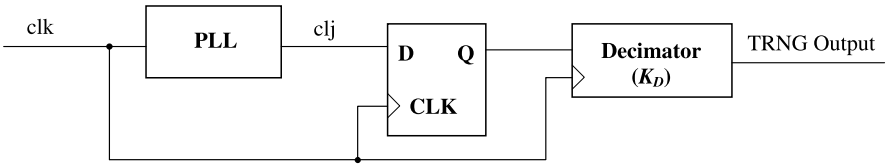


Fig. 5.9 Fischer and Drutarovsky's TRNG

where *CLK* is a reference clock signal. Parameters K_M and K_D are multiplication and division factors of the PLL respectively. The *CLJ* signal is sampled by the *D* flip-flop using a clock signal with a frequency F_{CLK}. K_D rising edges of *CLK* signal and $2K_M$ edges (rising and falling) of *CLJ* signal occur during one time period

$$T_Q = K_D T_{CLK} = K_M T_{CLJ}. \tag{5.13}$$

It was shown in [18] that if K_M and K_D are relative primes, the set of samples creates an equidistant set of values. The worst case distance between the two closest edges of *CLK* and *CLJ* during the period T_Q is given by

$$\mathrm{MAX}(\Delta T_{\min}) = \frac{T_{CLK}}{4K_M}\mathrm{GCD}(2K_M, K_D) = \frac{T_{CLJ}}{4K_D}\mathrm{GCD}(2K_M, K_D), \tag{5.14}$$

where GCD means Greatest Common Divisor. If K_M, K_D, and F_{CLJ} are chosen so that

$$\sigma_{jit} > \mathrm{MAX}(\Delta T_{\min}) \tag{5.15}$$

the sampling edge of *CLK* will fall at least once into the edge zone of *CLJ* (the edge zone means the time interval around the edge with a width smaller than σ_{jit}) during each period T_Q. For this reason, at least one of K_D samples will depend on random jitter. In order to remove the pseudo random pattern, K_D samples are XOR-ed in the decimator to give one output bit.

In [19], Fischer *et al.* showed that the sensitivity of the generator to jitter can be enhanced using two PLLs. If two PLLs are not available, a chain of delay elements can be used to increase the probability of overlapping of the *CLK* and *CLJ* edge zones [19]. Liu and McNeil proposed another PLL based TRNG that can be implemented in logic devices [38], but not in reconfigurable hardware.

The generator does not require post processing. Its output bit rate is relatively low, but remains constant. It uses very few logic resources, but the use of PLLs could be restrictive in some designs (some FPGAs contain only one or two PLLs). Because of the small logic area required by the generator, its power consumption could be very low, if PLL operation could be controlled (disabled). This is possible in most FPGAs featuring PLLs.

The generator is absolutely inner testable. Based on the simple principle of the generator (the use of two rationally related frequencies), in [46], the authors propose a simple stochastic model that specifies the dependence of the output (bias) on the jitter source statistical parameters (σ). The model can serve as the basis for the implementation of embedded tests in order to improve resistance against attacks.

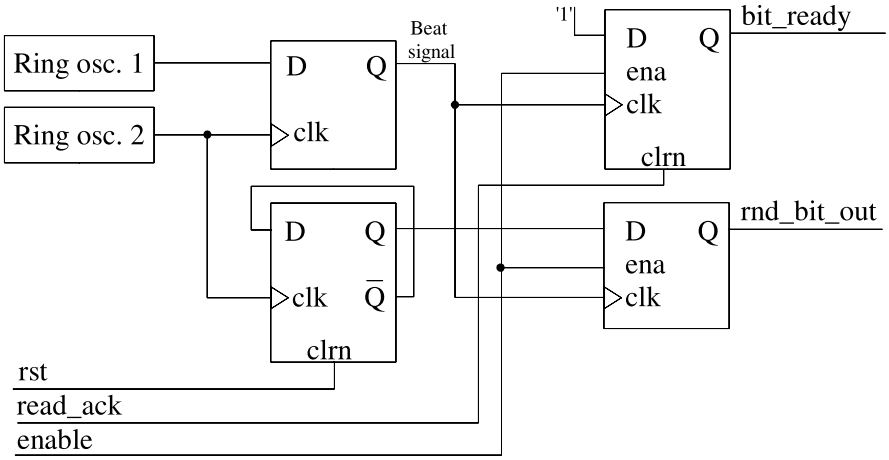


Fig. 5.10 Kohlbrenner and Gaj's Generator

The generator can be easily implemented in all FPGAs containing PLLs (most recent FPGA families, including Xilinx Virtex 5). The design can be fully automated and does not require any manual intervention. The only parameters that have to be adjusted to ensure correct operation of the generator are those of the PLL (K_M and K_D).

Kohlbrenner and Gaj's Generator Since the generator discussed in the previous section uses PLLs that are not available in all FPGAs, in [35], the authors propose a derivation of the same principle (see Fig. 5.10): they use a pair of ring oscillators oscillating at two very close frequencies (period difference is set to tens of ps). A *D* flip-flop is used to sample the first generated clock signal on the rising edge of the second one. The flip-flop output features a small frequency signal (a beat signal) with an unstable period related to the difference in frequency between the two clocks. In the following stage of the generator, a counter measures the unstable period of the beat signal. The least significant bit of the counter (implemented as a 1-bit T flip-flop) is sampled on the rising edge of the beat signal to give the next random bit of the generator.

It appears that the most critical part of this TRNG is the need for the precise setting of the two ring oscillator frequencies. The obtained frequencies have to be very close but not the same in order to obtain an appropriate beat signal. The frequency difference is directly proportional to the random output bit rate and indirectly proportional to the sensitivity on the jitter. In this way, if the two frequencies are very close, the generator will be able to extract very small jitter, but the beat signal will have very long period causing very small bit rate of the generator.

Due to process variations, the frequencies of the ring oscillators can differ up to 8% inside the chip and even more between chips [35]. Since the frequency difference is related to the sensitivity to the jitter, the generator could cease to work

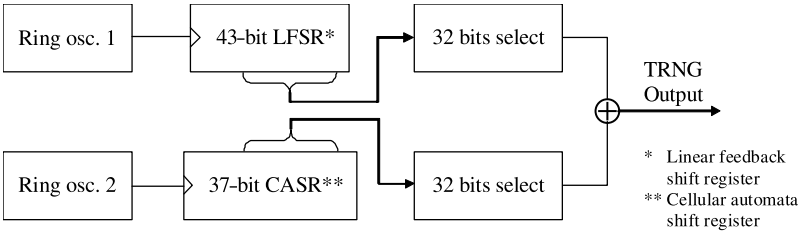


Fig. 5.11 Tkacik's TRNG

in some devices. In order to solve this problem, manual placement on a per-device basis may be needed. This is the main disadvantage of the proposed principle.

The generator is stateless and is thus absolutely inner testable (although to date, no test has been proposed). Since the structure of the generator is relatively simple, even if it is not provided in the original paper, a mathematical model should be feasible. The fact that the generator uses two frequencies generated using the same principle (ring oscillators) implemented in the same device should increase its robustness against active attacks.

The authors used the generator in Xilinx Virtex FPGA devices. They reported a bit rate of about 600 kbits/s. However, the output bit stream required post processing to reduce the bias to an acceptable level. The generator is potentially implementable in all FPGA families. It uses very few logic resources and could thus consume relatively little power.

5.5.4 Generators that Combine Pseudo and True Randomness

This class of generators combine true randomness obtained from free running oscillators (mostly ring oscillators) with pseudo randomness obtained from algorithmic logic structures such as Fibonacci, Galois and cellular automata shift registers (or rings). Since both sources of randomness share the same hardware, the generators are not inner testable and the entropy of the generated bit stream is very difficult or impossible to evaluate.

Tkacik's Generator The principle of random number generator proposed by Tkacik [51] is presented in Fig. 5.11. This TRNG is composed of two state machines: one linear feedback shift register (LFSR) and one cellular automata shift register (CASR), which are clocked by jittery clock signals from two independent free running oscillators. The registers have different lengths. The LFSR, which is based on a primitive polynomial, is 43 bits long, and the CASR is 37 bits long. Only 32 bits are selected from both. These 32-bit signals are permuted according to a fixed scheme (not published). The permutation reduces the correlation between the bits and is referred as 'site spacing' [31]. The two 32-bit vectors are then XOR-ed together in order to be sampled in the following 32-bit register, giving a 32-bit random number at its output.

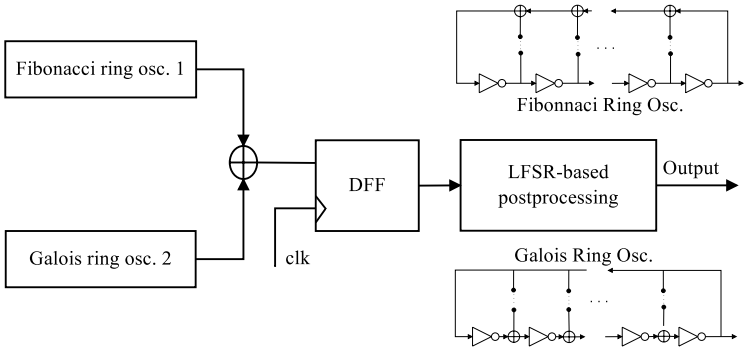


Fig. 5.12 Golic's TRNG

The oscillators that drive the shift registers never stop (free running) even when the generator is not in use. In this way, the LFSR and CASR are assumed to be in an undetermined state after a sufficiently long period (due to the drift in time of the two oscillator frequencies).

Tkacik's generator behaves according to its internal state, which is given by the current state of the LFSR and CASR. For this reason, when multiple successive random words are needed, the design of the TRNG requires that the application waits for a minimum period before sampling the next number. This allows the shift registers to complete their cycle at least twice between two successive random numbers [51].

The initial states of the registers that comprise the generator are not initialized at power up so that the initial state is (theoretically) unpredictable. Nevertheless, in [13], Dichtl showed that there could be serious security flaws and that (theoretically) the output of this generator is predictable. The flaws discussed in [13] are described only at a theoretical level with no practical application of the proposed attack. Since pseudo randomness and true randomness are mixed at the source, the generator does not require post processing in order to pass the NIST tests. The output speed is constant, but is limited by the fact that the generator requires a minimum sampling time that cannot be violated. However, the speed is still reasonably high.

The generator can be easily implemented in reconfigurable hardware and, depending on the choice of rings and sampling frequencies, can provide high throughput. Its design is not technology critical, so that it can be implemented without manual intervention.

Golic's Generator The principle of random number generator proposed by Golic [22, 23] is presented in Fig. 5.12. This TRNG is based on two ring oscillators with linear feedback obtained from generalized LFSRs (Fibonacci and Galois LFSR), while the shift registers are replaced by delay elements (inverters).

Just like in LFSRs, the linear feedback implemented inside both oscillators can be expressed using polynomials. However, since the shift registers are replaced by inverters, structures with linear feedback oscillate freely. Because of the intrinsic

variation in the time delay of each oscillator element, the internal state changes chaotically very rapidly and cannot be predicted. In [14], Dichtl and Golic estimate that each of the rings with linear feedback can give a random output value after a period as short as 25 ns.

In order to increase the robustness, the outputs of the two rings are mixed together in a XOR gate before being sampled in a *D* flip-flop. The final throughput is directly proportional to the sampling frequency, which should be chosen according to the spectrum of the signal present at the XOR gate (more precisely to the low frequency limit of the spectrum) in order to avoid output bit correlations.

The generator uses LFSR based post processing to reduce the bias of the digital noise present at the output of the XOR gate. The output bit rate can thus be very high and constant.

In [14], Dichtl and Golic propose an improvement of the original principle by restarting the oscillators from the same initial conditions. A novel sampling method that almost doubles the entropy rate is also proposed in this paper.

The entropy included in the digitized noise signal was shown to be very high [14]. However, there are three serious flaws in Golic's generator that have not been explained to date: (1) while the principle of the generator (the use of linear feedback rings generating a chaotic signal) suggests that the XOR gate output shouldn't be biased, the bias observed is surprisingly high; (2) the spectrum of the generated noise is not flat (white noise), but some frequencies are dominant; (3) some generator implementations stop running randomly.

The original version of the generator is inner testable, but not absolutely inner testable (it is not stateless). However, the enhanced version from [14] is absolutely inner testable. The behavior of the generator has not yet been sufficiently analyzed. Its complexity will probably cause obstacles in elaborating a mathematical model. The robustness of Golic's generator against attacks is difficult to estimate, but no attacks have been proposed up to now.

The generator uses relatively few logic resources (logic gates only) and the version that restarts the oscillators can be used in applications that require reduced power consumption. It is easy to implement in all FPGA families and its synthesis can be fully automated (no manual intervention needed).

5.5.5 Generators Using Open Delay Chain

This class of generators is made up of an open delay chain and enables high data rates. Two designs based on similar principles were proposed independently in 2007. One was implemented in ASIC [4], the other in FPGA [10]. Both designs use a delay line composed of a chain of *n* delay elements, while the delay of each element has to be smaller than the standard deviation of the jitter.

It is interesting to note that although both generators use very similar principles, the authors of the first one [4] consider extracting entropy from the clock jitter and those of the second one [10] from the metastability behavior of the latch. Although

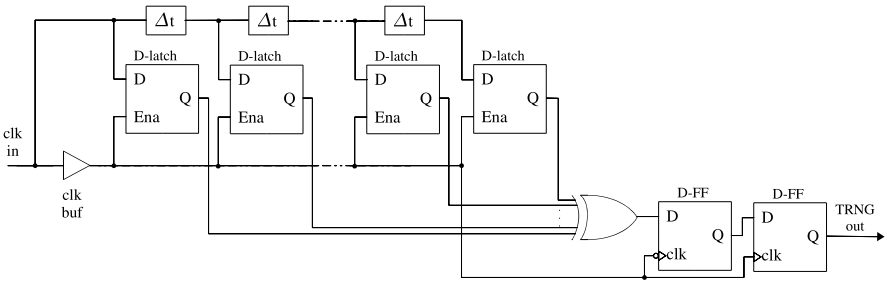


Fig. 5.13 Danger *et al.*'s TRNG

both generators have very similar structure and characteristics, we discuss only that of Danger *et al.*, since it was intended for FPGAs.

Danger *et al.*'s Generator The random number generator principle proposed by Danger *et al.* [10] is presented in Fig. 5.13. The input clock signal is split into two data paths: a data signal path and a clock signal path. The data signal enters the delay chain where the delay elements are merely parts of the same wire. Each delay element output is sampled by a D-latch built from an FPGA LUT. The n latches are enabled using the clock signal output from a global buffer and routed by a specific rail. Their outputs are XOR-ed together in order to capture changes in any one of them. The XOR gate output is resynchronized by DFFs to keep the state stable during one clock period.

The authors claim that by using the race between a signal and its delayed clone, it is possible to obtain a few latches along the delay chain to work near a metastable region. The latches in a metastable state converge towards a stable state depending on the noise level. If at least one latch is in a metastable state, the probability of 1's at the output will be $P \in]0, 1[$. However, the authors admit that the generator may accidentally get stuck at VSS or VDD because of the temperature or because of variations in the supply voltage caused unintentionally or by a malicious attacker. In order to counter this effect, in [11], they propose to dynamically adjust the delay ΔT between the clock signal and the data signal. The TRNG output probability is permanently calculated and exploited by a controller that drives a coarse delay chain built from FPGA carry or MUX primitives. This allows rapid delay chain calibration and switching towards the most appropriate metastable latch.

Because of discrete delay values, this TRNG requires post processing to reduce bias at the output. A simple von Neumann corrector is used because the reduction in the bit rate (at least by a factor of 4) is affordable as the original bit rate is as high as the clock frequency.

The output bit rate of the generator is potentially very high. The final value of the bit rate and its stability will depend on the post processing method used. However, it is very probable that for high clock frequencies, the subsequent bits will be correlated (because of the limited spectrum of the sampled jitter).

The generator is absolutely inner testable. It uses relatively few resources (logic gates and interconnect lines). Since the clock used to generate randomness can be

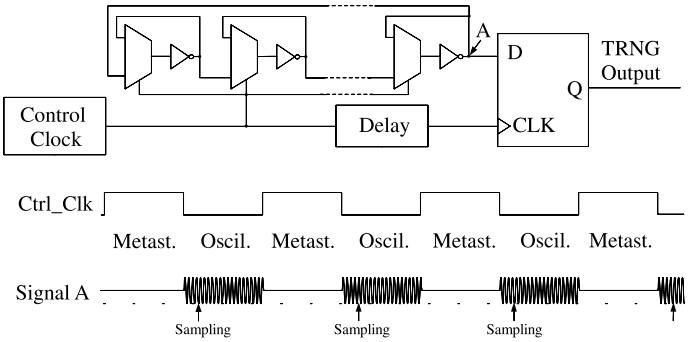


Fig. 5.14 Vasyltsov *et al.*'s TRNG

stopped, the generator can consume relatively little power. However, its design is strongly technology dependent (it depends on available routing resources). It is possible that it would not be feasible in some FPGA families (if delay elements cannot have sufficiently small delay). It is very probable that the design will need to be optimized for each FPGA family (depending on FPGA topology).

5.5.6 Generators Using Metastability

TRNG concepts based on metastability are much rarer in the literature than those based on noisy clock sampling or other concepts. Most of these generators are dedicated to ASICs [16, 30, 55]. Although some of them (e.g. that described in [55]) should be realizable in FPGAs, their implementation is not discussed. On the other hand, the generator published in [54] was intended especially for reconfigurable devices and was tested in Xilinx FPGAs.

Since FPGA manufacturers try to minimize metastability events in hardwired (dedicated) flip-flops, it seems that the best way to use metastability in FPGAs would be soft latches implemented in LUTs. This is the case of the generators described in [54, 55]. Both principles are similar to that described in [16] using a pair of cross connected inverters and multiplexers to switch the loops between oscillatory and bistable phases.

Vasyltsov *et al.*'s Generator The principle described in [55] uses a new type of ring oscillator working in two modes: metastability (entropy accumulation) mode and oscillations mode (see Fig. 5.14).

In the metastability mode, the loops of individual inverters are closed (using multiplexers). According to the authors, the output voltage of each inverter converges somewhere near a metastability level and remains there as long as the loop remains closed. Due to inherited thermal noise, the output voltage stochastically fluctuates around the metastable level. Since each inverter is disconnected from the others and

the threshold voltage is applied to its input, the inverters form a set of independent noise sources. After disconnecting the feedback loops of the inverters, the ring oscillator starts to oscillate. As claimed in the paper, its initial state depends on the entropy coming from stochastic fluctuations of each inverter.

Because no details about implementation are provided in the paper [55], it is quite difficult to assess the characteristics of the proposed generator. According to the information provided, we would expect that the generator could reach a relatively high and regular bit rate. However, the authors mention that the statistical parameters of the output bit stream varied with time and temperature. For this reason, the generator required stable temperature and voltage to function well. Another point that indicates some weaknesses of the generator is the fact that the output bit stream is strongly biased.

The proposed TRNG is absolutely inner testable, because it is reset regularly at the beginning of each entropy accumulation interval. The mathematical model has not been published up to now. The difficulty of designing such a model may be related to the complexity of the inverter's behavior in the metastable region. Since the original version of the generator has several implementation weaknesses, its robustness against attacks needs to be analyzed in detail.

The generator uses few logic resources and its power consumption is probably relatively low. Its feasibility in various FPGA technologies also needs to be validated.

Varchola and Drutarovsky's Generator The principle described in [54] extracts randomness from temporary oscillations when a bistable circuit is resolving a metastable event. The authors propose a new temporarily instable structure called Transition Effect Ring Oscillator (TERO), for which the oscillatory phases are introduced periodically on both the rising and falling edges of the control signal (see Fig. 5.15). The authors show that the number of oscillations depends on the intrinsic noise in logic cells. The oscillations are counted in a modulo 2 counter (T flip-flop) that stops counting at a random value reached when the oscillations die down.

This TRNG is absolutely inner testable because it is reset at regular intervals. The embedded tests can use the simple mathematical model presented in the paper to enhance the robustness of the generator against attacks. The TERO TRNG uses few logic resources and its power consumption is relatively low. Although it is feasible in FPGAs, its reliability depends on the period of the control clock and the number of oscillations. If the oscillations last longer than the half period of the control signal, the generator's output is constant. Note that number of oscillations depends mainly on the symmetry of the structure, but also on the size of the jitter and its accumulation (a perfectly symmetric oscillator without random jitter would oscillate indefinitely). For this reason, the control clock frequency would have to be adjusted for each device according to placement and routing, these depending on the device characteristics. When setting the frequency, the designer should also take into account the impact of temperature and voltage fluctuations.

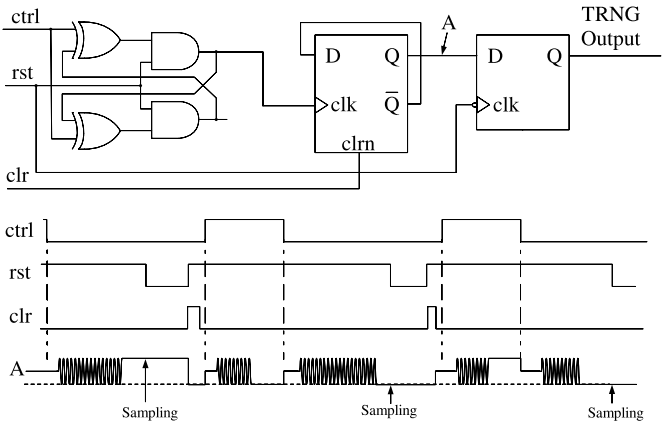


Fig. 5.15 Varchola and Drutarovsky's TRNG

5.5.7 Other TRNG Principles

In this section, we present principles that require special features that are rare in FPGAs or that are significantly topology dependent. For example, chaos based generators need some non-linear analog components for their implementation and RAM-based generators use randomly initialized embedded RAM blocks.

The theory of chaos, as a branch of the theory of non-linear dynamic systems, drew attention to a surprising fact: low-dimensional dynamic systems are capable of complex and unpredictable behavior, which is intuitively very promising for random number generation. Indeed, there are a large number of designs in this domain [7, 8, 15, 16, 39]. In [15], the authors use an on chip reconfigurable system from Cypress (featuring ADC) to generate randomness from chaos. No chaos based generator has been implemented in true FPGAs to date, but we expect that the production of an FPGA including ADC based on successive iterations (the kind of converter that enables chaotic behavior to be obtained) is only a question of time.

RAM-Based TRNGs It is well known that the RAM memory is initialized randomly when it is being powered up. The random contents of the FPGA RAM memory can be used during the initialization of each device to generate random numbers and/or to generate a device fingerprint such as a physical unclonable function (PUF) [29]. Note that while TRNGs and PUFs are often based on similar principles, their source of randomness is not the same: TRNGs are based on randomness originating from the exploitation of the device, while PUFs use randomness originating from the manufacturing process. However, random RAM contents cannot be found in all FPGAs, because the RAM memory in most volatile reconfigurable devices is initialized to zero or other user-defined value during device configuration.

In order to extend the use of RAM blocks as a source of randomness, two independent papers use written collisions in dual port embedded memories when writing

different data to the same memory location [25, 26]. The obtained bit rate can be very high, but the principle is not directly exploitable in all technologies. Because of the complexity of the process which depends on the device topology, it would probably be very difficile to build a stochastic model of the generator's behavior.

5.6 Conclusions

In this chapter, we presented various aspects of the design, implementation and evaluation of TRNGs. Basic generator principles were described, analyzed and compared. Although numerous principles can be exploited in FPGAs, many of them have disadvantages that limit their practical use in cryptographic applications in general, and especially in applications in which the generator is intended for the generation of confidential keys.

It is clear that the classical approach to TRNG design, which is limited to a statistical evaluation of the generator output, is no longer sufficient. We believe significant effort is necessary to better characterize existing sources of randomness in reconfigurable devices. Understanding the probability distribution and its characteristics for random processes taking place within the devices is difficult. However, the characterization of internal processes is crucial for the successful construction of stochastic models, which are indispensable for the reliable evaluation of entropy. Without a deep understanding of the entropy generation process, it is impossible to evaluate the robustness and security of the generator, and these are indispensable in cryptographic applications.

References

1. Bagini, V., Bucci, M.: A design of reliable true random number generator for cryptographic applications. In: 1st Workshop on Cryptographic Hardware and Embedded Systems—CHES 1999, Worcester, USA. Lecture Notes in Computer Science, vol. 1717, pp. 204–218. Springer, Berlin (1999)
2. Bock, H., Bucci, M., Luzzi, R.: An offset-compensated oscillator-based random bit source for security applications. Lecture Notes in Computer Science, pp. 268–281 (2004)
3. Bucci, M., Luzzi, R.: Design of testable random bit generators. In: Cryptographic Hardware and Embedded Systems—CHES 2005, Edinburgh, UK. Lecture Notes in Computer Science, vol. 3659, pp. 147–156. Springer, Berlin (2005)
4. Bucci, M., Luzzi, R., I.T. AG, Graz: A testable random bit generator based on a high resolution phase noise detection. In: Design and Diagnostics of Electronic Circuits and Systems, 2007. DDECS'07, pp. 1–5. IEEE (2007)
5. Bucci, M., Germani, L., Luzzi, R., Trifiletti, A., Varanonuovo, M.: A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC. IEEE Trans. Comput., 403–409 (2003)
6. Bucci, M., Giancane, L., Luzzi, R., Varanonuovo, M., Trifiletti, A., I.T. AG, Graz: A novel concept for stateless random bit generators in cryptographic applications. In: Proceedings. 2006 IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006, p. 4 (2006)
7. Callegari, S., Rovatti, R., Setti, G.: First direct implementation of a true random source on programmable hardware. Int. J. Circuit Theory Appl. **33**(1), 1–16 (2005)

8. Callegari, S., Rovatti, R., Setti, G.: Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. IEEE Trans. Signal Process. **53**(2), 793–805 (2005). See also IEEE Transactions on Acoustics, Speech, and Signal Processing
9. Colbourn, C.J., Dinitz, J.H., Stinson, D.R.: Applications of combinatorial designs to communications, cryptography, and networking. In: London Mathematical Society Lecture Note Series, pp. 37–100 (1999). Faculty of Mathematics and Dept. of Combinatorics and Optimization and University of Waterloo
10. Danger, J.L., Guilley, S., Hoogvorst, P.: Fast true random generator in FPGAs. In: IEEE MWSCAS/NEWCAS'07, Montréal, Canada, pp. 506–509 (2007)
11. Danger, J.-L., Guilley, S., Hoogvorst, P.: High speed true random number generator based on open loop structures in FPGAs. Microelectron. J. **40**(11), 1650–1656 (2009)
12. Davies, R.B.: Exclusive OR (XOR) and hardware random number generators. Online. Available at: http://webnz.com/robert/ (2002)
13. Dichtl, M.: How to predict the output of a hardware random number generator. In: Cryptographic Hardware and Embedded Systems—CHES 2003, Cologne, Germany. Lecture Notes in Computer Science, vol. 2779, pp. 181–188. Springer, Berlin (2003)
14. Dichtl, M., Golic, J.D.: High-speed true random number generation with logic gates only. In: Cryptographic Hardware and Embedded Systems—CHES 2007, Vienna, Austria. Lecture Notes in Computer Science, vol. 4727, pp. 45–61. Springer, Berlin (2007)
15. Drutarovsky, M., Galajda, P.: Chaos-based true random number generator embedded in a mixed-signal reconfigurable hardware. J. Electr. Eng.—Bratislava **57**(4), 218 (2006)
16. Epstein, M., Hars, L., Krasinski, R., Rosner, M., Zheng, H.: Design and implementation of a true random number generator based on digital circuit artifacts. In: Cryptographic Hardware and Embedded Systems—CHES 2003, Cologne, Germany. Lecture Notes in Computer Science, vol. 2779, pp. 152–165. Springer, Berlin (2003)
17. P. FIPS: 140-1: security requirements for cryptographic modules. National Institute of Standards and Technology, vol. 11 (1994)
18. Fischer, V., Drutarovsky, M.: True random number generator embedded in reconfigurable hardware. In: Cryptographic Hardware and Embedded Systems—CHES 2002, Redwood Shores, CA, USA. Lecture Notes in Computer Science, vol. 2523, pp. 415–430. Springer, Berlin (2002)
19. Fischer, V., Drutarovsky, M., Simka, M., Bochard, N.: High performance true random number generator in Altera Stratix FPLDs, FPL'04. Lecture Notes in Computer Science pp. 555–564 (2004)
20. Fischer, V., Bernard, F., Bochard, N., Varchola, M.: Enhancing security of ring oscillator-based RNG implemented in FPGA. In: Field-Programable Logic and Applications (FPL), pp. 245–250 (2008)
21. Fontaine, C.: Contribution à la recherche de fonctions booléennes hautement non linéaires, et au marquage d'images en vue de la protection des droits d'auteur. PhD thesis, Université Paris VI (1998). Online. Available at: http://www.irisa.fr/temics/staff/fontaine/these.html
22. Golic, J.D.: New paradigms for digital generation and post-processing of random data. Technical report, Cryptology ePrint Archive, Report 2004/254. Online. Available: http://eprint.iacr.org/2004/254.ps (2004)
23. Golic, J.D.: New methods for digital generation and postprocessing of random data. IEEE Trans. Comput., 1217–1229 (2006)
24. Gopalakrishnan, K., Stinson, D.R.: Applications of designs to cryptography. In: The CRC Handbook of Combinatorial Designs, pp. 549–557. CRC Press, Boca Raton (1996)
25. Güneysu, T.: True random number generation in block memories of reconfigurable devices. In: Proc. Int. Conf. on Field-Programmable Technology—FPT 2010, pp. 200–207. IEEE (2010)
26. Gyorfi, T., Cret, O., Suciu, A.: High performance true random number generator based on FPGA block rams. In: Proc. Int. Symposium on Parallel and Distributed Processing, pp. 1–8. IEEE (2009)
27. Hajimiri, A., Lee, T.H.: A general theory of phase noise in electrical oscillators. IEEE J. Solid-State Circuits **33**(2), 179–194 (1998)

28. Hajimiri, A., Limotyrakis, S., Lee, T.: Jitter and phase noise in ring oscillators. IEEE J. Solid-State Circuits **34**(6), 790–804 (1999)
29. Holcomb, D.E., Burleson, W.P., Fu, K.: Initial SRAM state as a fingerprint and source of true random numbers for RFID Tags. In: Proceedings of the Conference on RFID Security (2007)
30. Holleman, J., Otis, B., Bridges, S., Mitros, A., Diorio, C.: A 2.92 muW hardware random number generator. In: IEEE Proceedings of ESSCIRC (2006)
31. Hortensius, P.D., McLeod, R.D., Card, H.C.: Parallel random number generation for VLSI systems using cellular automata. IEEE Trans. Comput. **38**(10), 1466–1473 (1989)
32. Jun, B., Kocher, P.: The Intel random number generator. Cryptography Research Inc., White Paper (April 1999)
33. Killmann, W., Schindler, W.: AIS 31: Functionality classes and evaluation methodology for true (physical) random number generators, version 3.1. Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn (2001)
34. Killmann, W., Schindler, W.: A design for a physical RNG with robust entropy estimators. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2008. Lecture Notes in Computer Science, vol. 5154, pp. 146–163. Springer, Berlin (2008)
35. Kohlbrenner, P., Gaj, K.: An embedded true random number generator for FPGAs. In: Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, pp. 71–78 (2004). doi:10.1145/968280.968292
36. Kwok, S.H.M., Lam, E.Y.: FPGA-based high-speed true random number generator for cryptographic applications. In: TENCON 2006. 2006 IEEE Region 10 Conference, pp. 1–4 (2006)
37. Lee, D.J.: Modeling timing jitter in oscillators. Technical report, Mentor Graphics Corporation. Online. Available at: www.mentor.com/dsm (June 2001).
38. Liu, C., McNeill, J.: A digital-PLL-based true random number generator. In: Research in Microelectronics and Electronics, 2005 PhD, vol. 1 (2005)
39. Maher, D.P., Rance, R.J.: Random number generators founded on signal and information theory. In: Cryptographic Hardware and Embedded Systems. Lecture Notes in Computer Science, vol. 1717, pp. 219–230. Springer, Berlin (1999)
40. Marsaglia, G.: Diehard: battery of tests of randomness. Online. Available at: http://stat.fsu.edu/pub/diehard/ (1996)
41. Marsaglia, G.: The Marsaglia random number CDROM with the Diehard battery of tests of randomness. Online. Available at: http://www.csis.hku.hk/~diehard/ (1996)
42. Petrie, C.S., Connelly, J.A.: A noise-based IC random number generator for applications in cryptography. IEEE Trans. Circuits Syst. I, Fundam. Theory Appl. **47**(5), 615–621 (2000). See also IEEE Transactions on Circuits and Systems I: Regular Papers
43. Rukhin, A., Soto, J., Nechvatal, J., Smid, J., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22. Online. Available at: http://csrc.nist.gov/ (2001)
44. Schellekens, D., Preneel, B., Verbauwhede, I.: FPGA vendor agnostic true random number generator. In: Proc. 16th Int. Conf. Field Programmable Logic and Applications-FPL (2006)
45. Schindler, W., Killmann, W.: Evaluation criteria for true (physical) random number generators used in cryptographic applications. In: Cryptographic Hardware and Embedded Systems—CHES 2002, Redwood Shores, CA, USA. Lecture Notes in Computer Science, vol. 2523, pp. 431–449. Springer, Berlin (2003)
46. Simka, M., Drutarovsky, M., Fischer, V., Fayolle, J.: Model of a true random number generator aimed at cryptographic applications. In: Proceedings. 2006 IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006, p. 4 (2006)
47. Sunar, B., Martin, W.J., Stinson, D.R.: A provably secure true random number generator with built-in tolerance to active attacks. IEEE Trans. Comput., 109–119 (2007)
48. T. Technologies: Synchronous optical network (SONET) transport systems: common generic criteria. Technical report, GR-253-CORE (2000)
49. Tektronix: Understanding and characterizing timing jitter. Technical report, Tektronix. Online. Available at: www.tektronix.com/jitter (October 2003)

50. Tektronix: A guide to understanding and characterizing timing jitter. Technical report, Tektronix. Online. Available at: www.tektronix.com/jitter (2007)
51. Tkacik, T.E.: A hardware random number generator. In: Cryptographic Hardware and Embedded Systems—CHES 2002, Redwood Shores, CA, USA. Lecture Notes in Computer Science, vol. 2523, pp. 450–453. Springer, Berlin (2003)
52. Tsoi, K.H., Leung, K.H., Leong, P.H.W.: Compact FPGA-based true and pseudo random number generators. In: 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003, pp. 51–61 (2003)
53. Valtchanov, B., Aubert, A., Bernard, F., Fischer, V.: Modeling and observing the jitter in ring oscillators implemented in FPGAs. In: 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. pp. 1–6 (2008)
54. Varchola, M., Drutarovsky, M.: New high entropy element for FPGA based true random number generators. In: Mangard, S., Standaert, F.-X. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2010. Lecture Notes in Computer Science, vol. 6225, pp. 351–365. Springer, Berlin (2010)
55. Vasyltsov, I., Hambardzumyan, E., Kim, Y.-S., Karpinskyy, B.: Fast digital TRNG based on metastable ring oscillator. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2008. Lecture Notes in Computer Science, vol. 5154, pp. 164–180. Springer, Berlin (2008)
56. Wang, Y.H., Zhang, H.G., Shen, Z.D., Li, K.S.: Thermal noise random number generator based on SHA-2. In: Proceedings of 2005 International Conference on Machine Learning and Cybernetics (2005)
57. WAVECREST: Understanding jitter. Technical report, WAVECREST Corporation. Online. Available at: http://www.wavecrest.com/ (2001)
58. Wedge, S.W.: Predicting random jitter—exploring the current simulation techniques for predicting the noise in oscillator, clock, and timing circuits. IEEE Circuits Devices Mag. **22**(6), 31–38 (2006)
59. Yasuda, S., Tanamoto, T., Ohba, R., Abe, K., Nozaki, H., Fujita, S.: Physical random number generators for cryptographic application in mobile devices. In: Proceedings of the 31st European Solid-State Circuits Conference, 2005. ESSCIRC 2005, pp. 399–402 (2005)
60. Yoo, S.K., Sunar, B., Karakoyunlu, D., Birand, B.: A robust and practical random number generator. http://ece.wpi.edu/~sunar/preprints/rings.pdf (August 2007)

Chapter 6
Embedded Systems Security for FPGA

B. Badrignans, F. Devic, L. Torres, G. Sassatelli, and P. Benoit

Abstract The main goal of this chapter is to study FPGA devices in the field of secured applications. We mainly address data protection based on a well defined threat model. When dealing with FPGAs at the system level, two kinds of data are of paramount importance: bitstream and external memory. To cover these topics, we first review state of the art FPGA security mechanisms and good practices, followed by performance analysis achievable using hardware implementation of cryptographic algorithms in current FPGAs. We then tackle external memory protection and how FPGAs can provide an efficient solution. Next, we highlight security issues specific to FPGAs, bitstream replay attacks, for example, and suggest solutions to improve bitstream management security, focusing on secure remote updating of FPGA bitstreams. Finally we give the results of a concrete case, i.e., a platform based on an FPGA device. This last section provides both a practical and an industrial point of view that will enable readers to evaluate the pertinence of the solutions proposed.

6.1 Introduction and Objectives

Motivations to employ FPGAs (Field-Programmable Gate Array) in secure systems are multiple: hardware configuration can be updated all along system life-cycle, FPGAs can be finely configured to implement cryptographic functions efficiently, and security applications generally generate low sales volumes making FPGAs more attractive than ASICs (Application Specific Integrated Circuits). However ASICs often contain special features that are not available in all FPGAs. For instance, most current FPGAs do not include non-volatile memories that are useful in security applications (e.g., to store cryptographic keys). Moreover designers including FPGA devices in their design must not only consider in its threat analysis the applicability of attacks generally targeting ASICs and general purpose processors (e.g., memory tampering) but must also explore threats specific to FPGAs. For instance, an adversary tampering with the FPGA configuration file, a.k.a. the bitstream, can modify

B. Badrignans (✉)
Netheos, Montpellier, France
e-mail: b.badrignans@netheos.net

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_6, © Springer Science+Business Media B.V. 2011

functions implemented inside the FPGA user logic and thereby impact the behavior of the FPGA-based system. The main goal of this chapter is to explore threats potentially impacting FPGA-based systems. We address data protection based on a well defined threat model where the adversary has physical access to the FPGA-based system.

We mainly address data/code protection since two kinds of data—bitstream and external memory—are of paramount importance when dealing with FPGAs at the system level. In order to highlight the importance of the threats discussed in this chapter, we identified three security-sensitive applications potentially including FPGAs in their designs.

Physically Inaccessible Systems Some systems like satellite or space craft are intrinsically protected against physical attacks since inaccessible during their deployment in space. Devices can also be made inaccessible using secure rooms or strong-box. Due to their peculiar location we can consider that they are protected against any attack requiring physical access to the device. However the cost of this system as well as their potential strategic importance (e.g., in military and communication satellites) emphasis the requirement to make them highly secure against remote attacks.

Personal Hardware Security Modules—HSM Hardware security modules are devices dedicated to provide a high level of security which generally cannot be achieved using general purpose computers. They often offer ways to physically protect cryptographic keys or sensitive data. These systems are used in highly secure applications such as in-line banking applications, ATMs transactions, and companies network infrastructures. Since the appearance of smart card for payment and money withdrawal, most individual owns an increasing number of security modules (e.g., biometric passport, mobile phone SIM card, French health insurance card).

Smart cards do not embed programmable logic yet (even though the growing market of multi-applications cards might encourage such features in a near future [25]) but they are not the only personal secure devices available. For instance the smart drive provided by the French company Bull embeds an FPGA device in charge of cryptographic and key management operations. This device, called *Globull*, is a secure USB hard disk drive protected by a user PIN (Personal Identification Number). It provides also cryptographic services like smart cards, such as on-board RSA key generation, RSA signature and encryption. Therefore we can reasonably imagine that in coming years most individuals traveling with sensitive data will have advanced personal security devices embedding FPGA devices. Applications can be: mobile disk encryption, e-mail protection, secure key management, VPN access.

Therefore we can reasonably assume that in the near future most individuals who travel with sensitive data will own advanced personal security devices that can take embedded FPGA devices. Some applications are mobile disk encryption, e-mail protection, secure key management, or VPN access. Whatever the application, personal security devices have their own constraints. Firstly, most devices assume that attackers will never access them when they are unlocked by the user. For instance,

laptop disk encryption is ineffective if someone steals it when its disk is unlocked. In addition, the devices can reasonably trust their owner, after all the information that these devices protect belong to the user.

Set-Top Boxes/Video Game Console Set-top boxes are devices generally rented out to customers by ISP (Internet Service Provider) or pay-TV providers. Since these systems provide access to protected multimedia content they must embed some security features to enforce digital right management. Their environments are dramatically different from previous examples of applications. First such systems are not shielded or physically protected and the user is potentially the adversary. Therefore, the adversary has physical access and is not limited in time to carry out his/her attack. Moreover he often benefits from a community of attackers or security researchers [42] sharing their technical knowledge and discoveries online.

In order to address security concerns related to these possible scenarios, this chapter is divided into three main parts each of which addresses one of the three key points involved in dealing with security and FPGAs. The first point is related to protecting data and code processed by FPGAs. We present a widely recognized threat model and existing efforts to protect a system against these threats, and then several possible solutions. These approaches leverage applications and FPGA characteristics to reduce the cost of security. The benefit is a strong optimization component in terms of memory overhead and performance. The second point is related to reconfiguration management of targeted architectures. This point is very sensitive as FPGAs offer considerable flexibility through dynamic reconfiguration. This feature can not only be used to perform upgrades and bug fixing, but also allows for reactions in the case of an attack. However, if not handled correctly, this strong advantage can become a security breach. After describing potential threats that use the reconfiguration link, we detail current efforts to counter them. Then we propose an original technique to perform remote partial reconfiguration. Based on such technology, a whole protocol and architecture were designed to perform secure reconfiguration. The third point is defining a secure platform. We thoroughly analyze what is required to obtain a fully secure FPGA based on previous solutions. A prototype of this platform has been designed and offers interesting features to target, including gigabit network encryption, SSL accelerator or offload engine, VPN accelerator, Hardware Security Module with high performances.

6.2 Definitions—Glossary

In this chapter, the following vocabulary is used to distinguish the different internal logic types in FPGA devices:

- **static logic** is the static part of the FPGA; the designer cannot modify its architecture. FPGA devices often embed general purpose processor in the static logic.
- **user logic** is the configurable logic embedded in FPGA devices. The user logic may include LUTs, RAM blocks, hardware multipliers, DSP blocks or switch matrix for routing signals.

- **configuration logic** is the part of the static logic in charge of loading the bitstream into the user logic, it is typically composed of a JTAG chain or any configuration port and of the bitstream decryption engine (if any).

In the following section we distinguish four stakeholders involved in an FPGA-based system development life cycle:

- The **FPGA vendors** are the companies designing, producing and providing FPGA chips (e.g., Xilinx, Altera or Actel).
- **IP designers** provide reusable units of hardware units often delivered as netlist or Hardware Description Language (HDL) codes. IP cores may be encryption engines, general purpose processors or memory interfaces.
- The **System Designer** (SD) assembles IPs provided by different IP designers to produce the final bitstream which configures the FPGA device. In case of complex FPGA-based systems, we assume in this book that the SD is also responsible to produce the platform potentially composed of different types of circuit (e.g., General purpose processors, ASICs or FPGAs).
- The **system owner** is the end user who exploits the system, the SDs and IP designers do not necessarily trust owners.

We distinguish three types of FPGA chips with respect to their configurability properties:

Current FPGAs can be classified in three different types:

- **Anti-fuse** FPGAs are historically the first non-volatile FPGAs. Each configuration point is controlled by an anti-fuse element. Configuration is fixed and cannot be changed after programming. Actel is the leader in this market.
- **Flash-based** FPGA configuration sites are controlled by a Flash transistor, they are reconfigurable but non-volatile.
- The last type is **SRAM based** FPGAs or volatile FPGAs are composed of SRAM memory cells and, therefore, cannot keep their configuration when power is down. An external non-volatile memory is generally required to store the bitstream. Altera and Xilinx are the leading companies in the market of SRAM FPGAs, offering low-cost as well as high-performance SRAM-based FPGA devices.

FPGAs can be configured in two main ways:

Static Reconfiguration is the classical ability of FPGA devices to be reconfigured when powered down. This feature is useful for cryptography and security-sensitive applications. Cryptographic algorithms have a limited lifetime, the system designer can provide new algorithms during the exploitation of the system. New attacks can also be discovered and again the system designer can provide a more robust version of the system.

Dynamic Reconfiguration is the ability of FPGAs to be reconfigured at runtime. Xilinx has been supplying tools and mechanisms for this purpose for many years. This feature allows the system to swap logic blocks thus allowing many applications. However, the process also introduces security issues since portions of the bitstream

need to be secured. The system designer may encrypt and authenticate these pieces of hardware in order to avoid a malicious bitstream being loaded. This has to be done inside the user logic since the SD is in charge of partial reconfiguration through an internal port located in the user logic, called ICAP for Internal Configuration Access Port. Partial reconfiguration security is not addressed in this book since it is not applicable on all FPGAs, and also because the concepts developed to secure classical bitstreams can be used for partial reconfiguration.

6.3 Confidentiality and Integrity of Data Processed by Reconfigurable Platforms

Data processed by general purpose processors embedded in the static logic of FPGAs or by IPs implemented in the user logic are commonly stored in off-chip memories. In applications where physical adversaries are considered (e.g., set top box), an attacker can retrieve data transiting on the bus, thereby challenging data confidentiality, or tamper with them, thereby challenging data integrity. In this section we first describe the passive and active attacks allowing an adversary to retrieve or tamper with data transiting between the FPGA chip and the memory. Then we describe potential countermeasures to this security issue and solutions we recently proposed.

6.3.1 Threat Analysis

Confidentiality and integrity of data stored in off-chip memory is usually a security concern when attackers with physical access to the device are considered (e.g., in application like set-top box). However, the main trust assumption made is that the processor chip is resistant to all physical attacks and is thus trusted. Moreover, the cryptographic engine required for encryption and authentication are assumed resistant to side channel attacks. We consider the adversary has full control of the data stored in memory and transiting on the bus between the FPGA chip and the memory. We consider two kinds of physical attacks an adversary can carry out: passive and active attacks. Passive attacks consist in probing the bus to retrieve the memory content. Such attacks challenge the confidentiality of data in memory.

In an active attack, the adversary corrupts the data residing in memory or transiting over the bus; this corruption may be considered as data injection since a new value is created. Figure 6.1 gives the example of an attacked device where an adversary connects his own (malicious) memory to the targeted platform via the off chip bus.

We distinguish between three classes of active attacks, defined with respect to how the adversary chooses the inserted data. Figure 6.2 depicts the three active attacks; below, we provide a detailed description of each one based on the attack framework in Fig. 6.1:

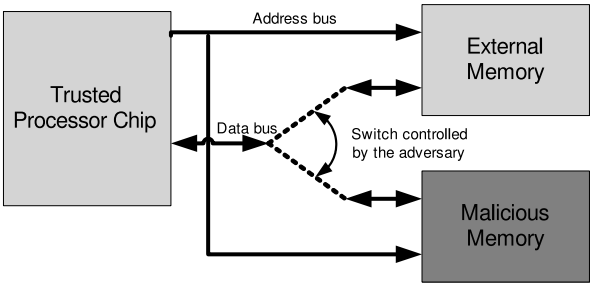


Fig. 6.1 An example of a framework of attack targeting the external memory of a computing platform

1. **Spoofing attacks**: the adversary exchanges an existing memory block with an arbitrary fake one (Fig. 6.2-a, the block defined by the adversary is stored in the malicious memory, the adversary activates the switch command when he wants to force the processor chip to use the spoofed memory block).
2. **Splicing or relocation attacks**: the attacker replaces a memory block at address A with a block at address B, where $A \neq B$. Such an attack can be considered

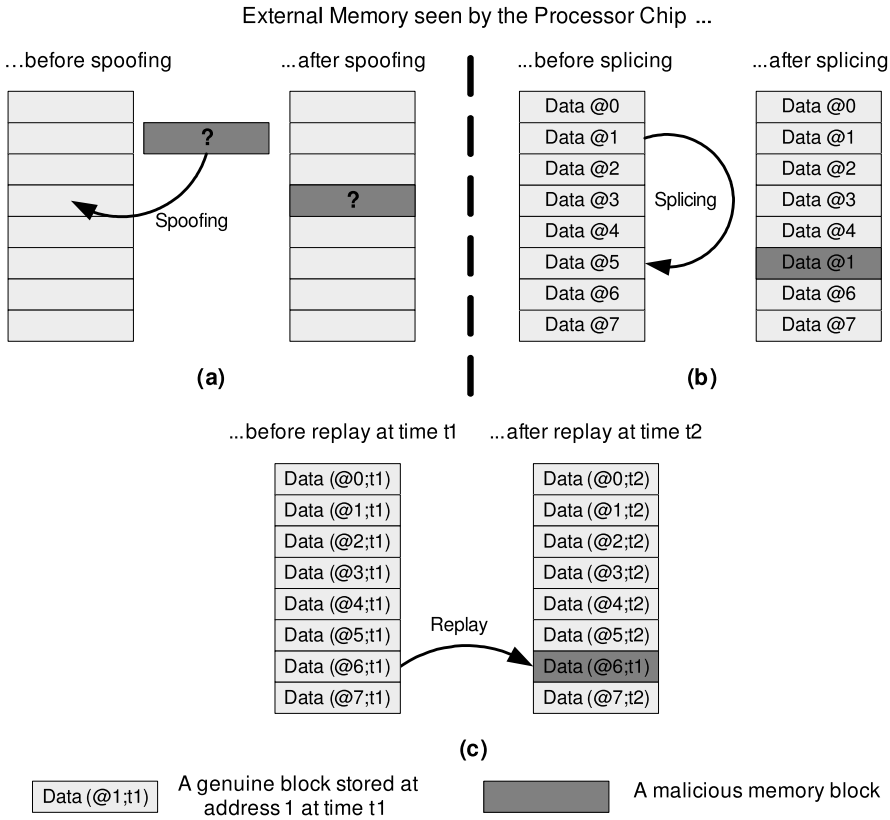


Fig. 6.2 Three kinds of active attacks: (**a**) spoofing, (**b**) splicing and (**c**) replay

as a spatial permutation of memory blocks (Fig. 6.2-b: the adversary stores the content of the block at address 1 in the genuine memory at address 5 in the malicious memory. When the processor requests the data at address 5, the adversary activates the switch command so the processor reads the malicious memory. As a result, the processor reads the data at address 1).

3. **Replay attacks**: a memory block located at a given address is recorded and inserted at the same address at a later point in time; by doing so, the value of the current block is replaced by an older one. Such an attack can be considered as a temporal permutation of a memory block, for a specific memory location (Fig. 6.2-c: at time t1, the adversary stores the content of the block at address 6 in the genuine memory at address 6 in the malicious memory. At time t2, the memory location at address 6 has been updated in the genuine memory but the adversary does not perform this update in the malicious memory. The adversary activates the malicious memory when the processor requests the data at address 6, thus forcing it to read the old value stored at address 6).

6.3.2 State of the Art

Two distinct strategies are described in the state of the art to thwart the active attacks described in our threat model. Each strategy is based on different authentication primitives, namely a cryptographic hash function and a message authentication code (MAC) function. In this section, we first describe how these primitives allow for memory authentication and how they should be integrated in tree structures in order to avoid excessive overheads in on-chip memory.

6.3.2.1 Authentication Primitives for Memory Authentication

Hash Functions The first strategy (Fig. 6.3-a) that enables memory authentication consists in storing on-chip a hash value for each memory block stored off-chip (write operations). The integrity of read operations is checked by re-computing a hash over the loaded block and by then comparing the resulting hash with the on-chip hash fingerprinting the off-chip memory location. The on-chip hash is stored in the tamper resistant area, i.e., the processor chip, and is thus inaccessible to adversaries. Therefore, spoofing, splicing and replay are detected if a mismatch occurs in the hash comparison. However, this solution may have an unaffordable on-chip memory cost: by considering the common strategy [17, 22, 38] of computing a fingerprint per cache line and assuming 128-bit hashes and 512-bit cache lines, the overhead will be 25% of the memory space to be protect.

MAC Functions In the second approach (Fig. 6.3-b), the authentication engine embedded on-chip computes a MAC for every data block it writes in the physical memory. The key used in the MAC computation is securely stored on the trusted

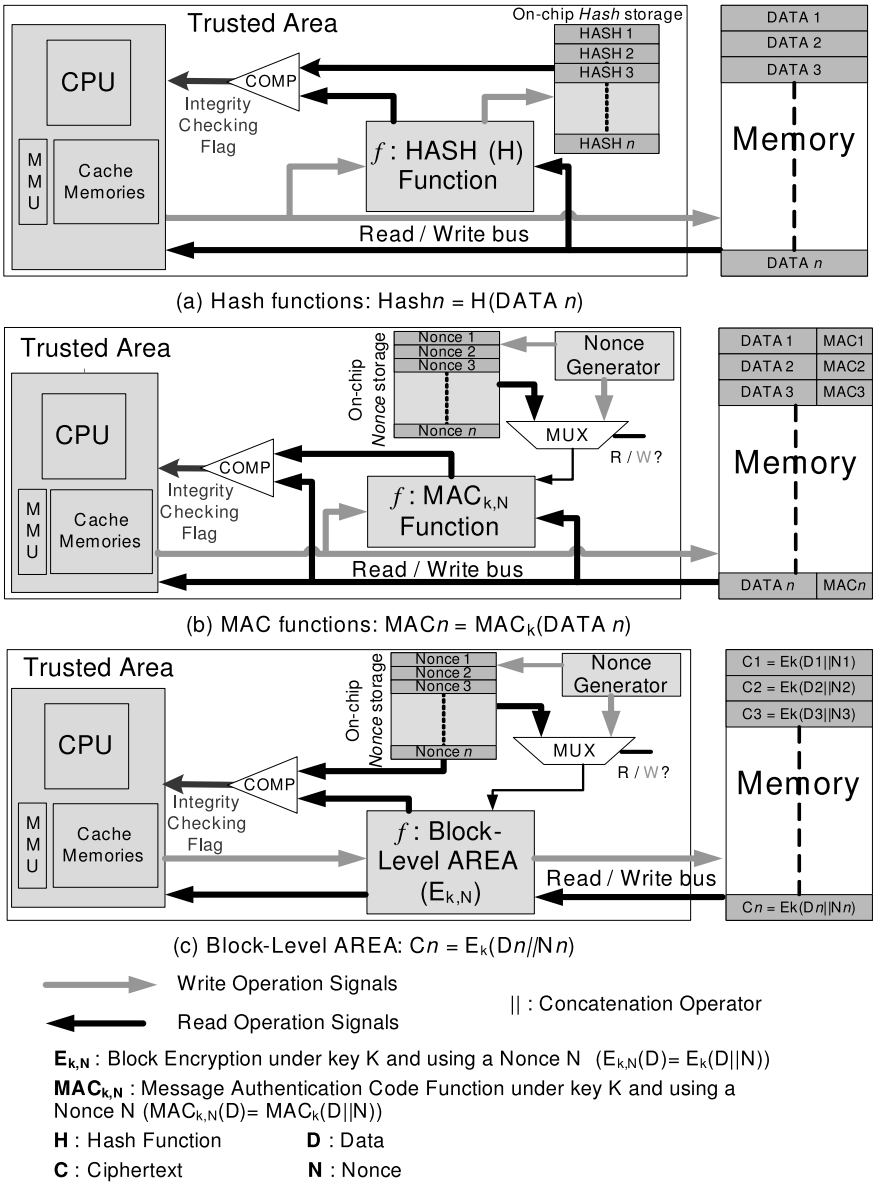


Fig. 6.3 Authentication primitives for memory integrity checking

processor chip such that only the on-chip authentication engine itself is able to compute valid MACs. As a result, the MACs can be stored in untrusted memory because the attacker is unable to compute a valid MAC over a corrupted data block. In addition to the data contained by the block, the pre-image of the MAC function contains a nonce. This enables protection against splicing and replay attacks.

The nonce precludes an attacker from passing off a data block at address A, along with the associated MAC, as a valid (data block, MAC) pair for address B, where $A \neq B$. It also prevents the replay of a (data block, MAC) pair by distinguishing two pairs related to the same address, but written in memory at different points in time. In read operations, the processor loads the data to be read and its corresponding MAC from the physical memory. It checks the integrity of the loaded block by first re-computing a MAC over this block and a copy of the nonce used in the write operation and by then comparing the result with the fetched MAC. However, to ensure the resistance to replay and splicing, the nonce used for MAC re-computation must be genuine. A naive solution to assure this requirement is to store them in the trusted and tamper-evident area, the processor chip. The related on-chip memory overhead is 12.5% in the case of computing a MAC per 512-bit cache line using 64-bit nonces.

6.3.2.2 Integrity Trees

In the previous section, we presented two authentication primitives that can prevent the active attacks described in our threat model. These primitives require storage of reference values—i.e., hashes or nonces—on-chip to thwart replay attacks. They do provide memory authentication but only at a high cost in terms of on-chip memory. If we consider a realistic case of 1 GB of RAM memory, the hash and MAC (with nonce) solutions require respectively at least 256 MB and 128 MB of on-chip memory. These on-chip memory requirements are clearly not affordable even for high end processors. It is thus necessary to "securely" store these reference values off-chip. By securely, we mean that we must be able to ensure their integrity to preclude attacks on the reference values themselves. Several authors suggest applying the authentication primitives recursively on the references. By doing so, a tree structure is formed and only the root of the tree (the reference value obtained in the last iteration of the recursion) needs to be stored on the processor chip, the trusted area. There are two existing tree techniques (in addition to those described in this work):

1. Merkle Tree [10] uses hash functions and is historically the first integrity tree. It was originally introduced by Merkle [30] to authenticate digital signatures and adapted for integrity checking of memory content by Blum et al. [10].
2. PAT (parallelizable authentication tree) [24] overcomes the issue of non-parallelizability of the tree update procedure by using a MAC function.

General Model of Integrity Tree The common philosophy behind integrity trees is splitting the memory space to be protected into M equal size blocks that are the leaf nodes of the balanced A-ary integrity tree (Fig. 6.4). The remaining tree levels are created by recursively applying the authentication primitive f over A-sized groups of memory blocks, until the procedure yields a single node called the root of the tree. The arity of the constructed tree is thus defined by the number of children A a tree node has. The root reflects the current state of the entire memory space; making the root tamper-resistant thus ensures tampering with the memory

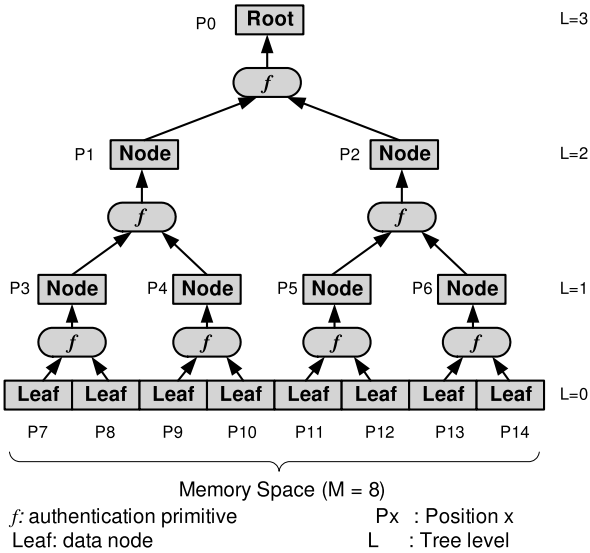


Fig. 6.4 General model of 2-ary integrity tree

space can be detected. How the root is made tamper-resistant depends on the nature of f and is detailed below. Note that the number of checks of the verification of the integrity of one leaf node required depends on the number of iterations of f and thus on the number of blocks M in the memory space. The number of checks corresponds to the number of tree levels L defined by: $L = \lg M$.

6.3.2.3 Execute-Only Memory (*XOM*)

XOM [28] is a security solution from Stanford University. The *XOM* approach, which provides memory protection, is based on a complex key management. The main *XOM* features are data ciphering, data hashing, data partitioning, interruption and context switching protection. Figures 6.5 and 6.6 provide an overview of the *XOM* architecture and mechanisms. All the security primitives are included in the trusted zone. The only security information that is not in the trusted zone are the session keys. That is why *XOM* owns a complex key management to guarantee a secure architecture.

The first version of *XOM* [28] is known to have security holes, like no protection against replay attacks. In [45], the authors extended their proposal and replaced the AES-based ciphering scheme with a system based on OTP to guarantee protection against replay attacks and also to increase the performances of the system. Concerning the global security level of the *XOM* architecture, the attack possibilities are fully dependent on the integrity checking capabilities. To succeed, attackers must be able to pass through the integrity check in order to execute their own program or use their own data. They may exploit some collisions in the hash algorithm used. For example, with MD5 the signature is 128 bits long. If attackers wish to attack the system, they need to find two inputs that will produce the same result with MD5.

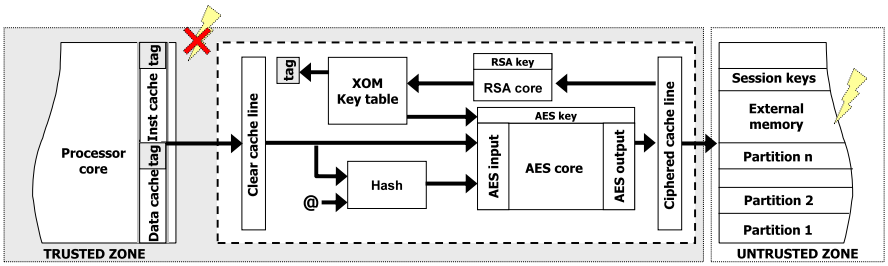


Fig. 6.5 XOM architecture for write request

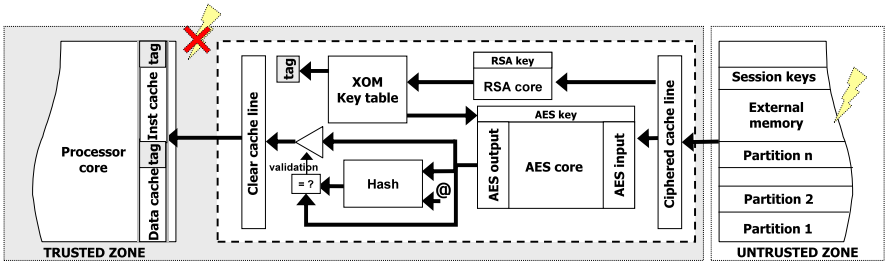


Fig. 6.6 XOM architecture for read request

So they have one chance out of 2^{128} to get the same result. The security level of the *XOM* mainly depends on the hash algorithm used, because SHA-1 could be used for integrity checking. In this case the signature would be 160 bits long and the probability of success would be one out of 2^{160}.

6.3.2.4 AEGIS

AEGIS [37, 40] is an additional memory security solution from Massachusetts Institute of Technology. The confidentiality in the AEGIS solution relies on OTP encryption [39]. This encryption method typically has a small impact on memory latency at the cost of memory space. The solution used by AEGIS for integrity checking is called cached hash tree. This hashing approach is similar to a Merkle tree [30] but to increase the efficiency of the method, some hash tree nodes are stored in a cache memory included in the secure zone. The advantage is that instead of computing all the tree nodes to the root, the system only needs to compute the values until one value from the secured memory is reached. The only weakness in this solution is architecture performance. Architecture performance is fully dependent on the size of the cache memory used to store the secured nodes. Architecture performance also depends on the algorithm used for hashing. As mentioned above, SHA-1 computation requires 80 cycles and MD5 only 64. AEGIS thus appears to be a very complete solution to protect memory and program. The overhead is high in several domains. The silicon area increased by 1.9 [40]. The CPU core is the part that is the

most affected by this overhead. Moreover, all the logic needed to control the specific mechanisms contributes to increasing the area (OTP core and hash core). The global architecture performances depend on parameters like the size of the protected memory and of the cache memory. For security concerns, like XOM, AEGIS depends on the integrity checking capabilities of the hash algorithm used for the Merkle tree. In [40], the authors use SHA-1 which leads to a 160 bit signature. This means that the likelihood of a successful attack is one out of 2^{160}.

6.3.3 Proposed Memory Authentication Techniques

In this section, we describe memory authentication techniques proposed recently. We first present a new authenticated encryption mode, the Block Level AREA (Added Redundancy Explicit Authentication), which provides data integrity and confidentiality and a cryptographic engine, called PE-ICE (Parallelized Encryption and Integrity Checking Engine), based on this mode. We then show that the Block Level AREA, like the authentication primitives described in Sect. 6.3.2, have to be integrated into a tree structure called TEC-Tree (Tamper-Evident Counter Tree), in order to avoid excessive overheads in on-chip memory. In the last part of this section, we present another approach named AES-TASC (Time Address Segment Cipher) that allows the use of a security policy to reduce the overhead due to security mechanisms.

6.3.3.1 Block-Level AREA and PE-ICE

Block-Level AREA Block-Level AREA [15, 17] (Fig. 6.3-c) leverages the diffusion property of block encryption to add an integrity checking capability to this type of encryption algorithm. To do so, the AREA (Added Redundancy Explicit Authentication [20]) technique is applied at the block level:

1. Redundant data (an n-bit nonce N) is concatenated to the data D we want to authenticate to form a plaintext block P (where $P = D||N$), ECB (Electronic CodeBook) encryption is performed to generate ciphertext C.
2. Integrity verification is done by the receiver who decrypts the ciphertext block C' to generate plaintext block P', and checks the n-bit redundancy in P', i.e., assuming $P' = (D'||N')$, verifies whether $N = N'$.

Thus, in a *memory write*, the on-chip authentication engine appends an n-bit nonce to the data to be written to memory, encrypts the resulting plaintext block and then writes the ciphertext to memory. The encryption is performed using a key securely stored in the processor chip. In *read operations*, the authentication engine decrypts the block it fetches from memory and checks its integrity by verifying that the last n bits of the resulting plaintext block are equal to the nonce that was inserted during encryption (in the write of the corresponding data). [15, 17] propose a system

on chip (SoC) implementation of this technique for embedded systems. They show that this engine can efficiently protect read only (RO) data of an application (e.g., its code) because RO data are not sensitive to replay attacks; therefore the address of each memory block can be efficiently used as a nonce.[1] However, for Read/Write (RW) data (e.g. stack data) the address is not sufficient to distinguish two data writes at the same address but at two different points in time. To recover the nonce in a read operation while ensuring its integrity, [15, 17] propose storing the nonce on chip. They evaluate the corresponding overhead at between 25% and 50% depending on the block encryption algorithm implemented.

PE-ICE A Parallelized Encryption and Integrity Checking Engine, PE-ICE was designed [15, 17], based on the block level AREA technique to encrypt and authenticate off-chip memory. However, to avoid re-encryption of the whole memory when the nonce reaches its limit (e.g., a counter that rolls over), we propose to replace it with the chunk address concatenated with a random number. For each memory block processed by PE-ICE, a copy of the enrolled random value is kept on chip to make it tamper resistant and secret. In the following, a *PE-ICE configuration* is defined as an implementation of PE-ICE with a given block cipher. A PE-ICE configuration is denoted PE-ICE-*bw* where *bw* is the bit width of the block processed by the underlying block cipher. In this section we first describe a PE-ICE configuration. Then we evaluate the performances of several PE-ICE configurations at runtime.

PE-ICE-160—A PE-ICE Configuration. The Rijndael algorithm is the block cipher that won the NIST contest for a new block encryption standard. The related standard is called AES [2] (Advanced Encryption Standard). AES processes 128-bit blocks and enrolls 128, 192 or 256-bit keys. However, the original Rijndael [11] block cipher supports any key and block sizes that are a multiple of 32 between 128 and 256. This leads to several configurations for PE-ICE based on this block cipher. We studied three of them, PE-ICE-128, PE-ICE-160 and PE-ICE-192, that use the Rijndael algorithm processing respectively 128-bit (AES), 160-bit (Rijn-160) and 192-bit (Rijn-192) blocks. For the sake of clarity, we only detail PE-ICE-160 configuration in this chapter; for a description of the other configurations, the reader is referred to [15].

PE-ICE shifts the physical address by inserting tags between payloads. This shift must be transparent for the CPU, thus PE-ICE handles the address translation (Fig. 6.7).

Memory Consumption The amount of memory consumed by PE-ICE depends on the tag storage of the off chip memory and on the storage of the reference random values for the on chip memory. The off chip memory overhead is defined by the ratio between the tag and the payload bit widths. For PE-ICE-160, the off chip memory overhead is 25%. The on chip memory overhead is defined by the ratio of the bit-length of a random value used to protect an RW chunk against replay to

[1] Note that the choice of the data address as nonce also prevents spoofing and splicing attacks on RO data when MAC functions are used as authentication primitives.

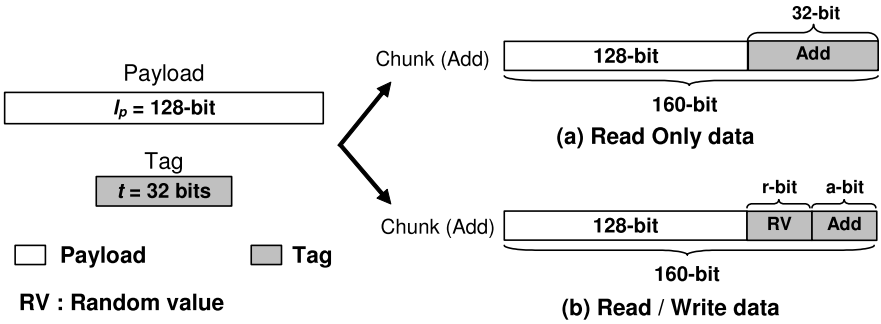


Fig. 6.7 Layout of a PE-ICE-160 chunk before encryption

the bit length of the corresponding protected payload. For PE-ICE-160, the on chip memory overhead is respectively 6.25% and 25% depending if it is 8 bits or 32 bits long. As we show in Sect. 6.3.3.2, we proposed in [15, 18] a scheme to reduce this overhead.

The cost of data authentication in PE-ICE can be evaluated by its overhead compared to AES-ECB encryption. On average this cost is 22%. This latency overhead is partially due to the increase in the intrinsic latency of the underlying block cipher. The hardware cost of PE-ICE-160 and of the AES-ECB is approximately 80 Kgates. At no additional hardware cost and with a low latency overhead, we showed that PE-ICE:

1. Strengthens AES-ECB encryption—the tag inserted before encryption prevents an adversary from detecting when the same data is transferred twice by monitoring bus transactions.
2. Provides data authentication in addition to data confidentiality.

Performance Evaluation *Results* Eight benchmarks [1] designed for embedded systems were used in this evaluation, running on an ARM processor core. The simulation results for the base platform serve as the reference and are shown in IPC (instructions per cycle) in Fig. 6.8 for two different data cache and instruction cache sizes (4 KB and 128 KB). We observed that the performance slowdown was mainly related to the data cache miss rate, see Fig. 6.9.

Figure 6.10 shows the simulation results of the platforms emulating the AES-ECB engine, PE-ICE-128, PE-ICE-160 and PE-ICE-192, in IPC normalized to the performance of the base platform. The AES-ECB engine chart clearly shows that the overhead of PE-ICE is mainly due to encryption; in the worst case it is 50% (CJPEG–4 KB) and 31.5% and of 14.3% on average for a 4 KB and 128 KB data cache, respectively. This quite significant timing performance cost can be dramatically reduced by using a wider processor-memory bus (e.g. 64 bits) and by running the encryption algorithm at its maximum frequency. We evaluated the implementation of PE-ICE with several block ciphers and showed that it provides data integrity in addition to data confidentiality with negligible hardware cost and performance

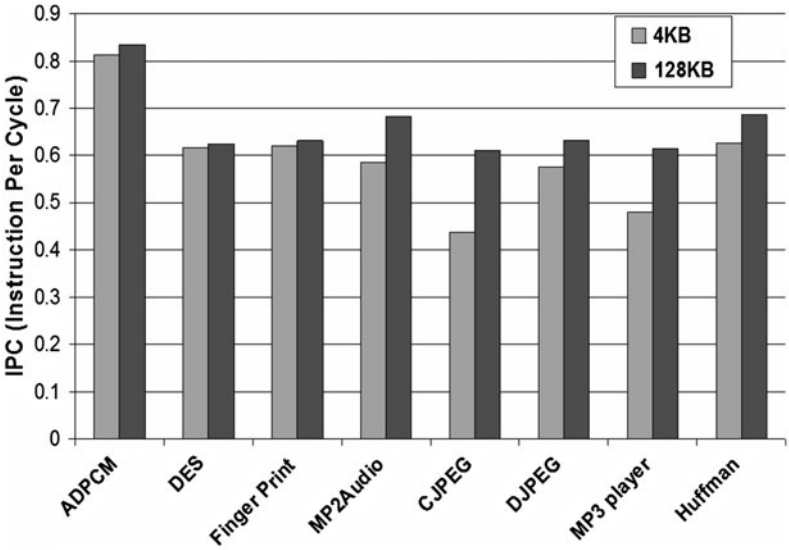


Fig. 6.8 Simulation results of the base platform for two different data cache sizes (4 KB and 128 KB) and two different instruction cache sizes (4 KB and 128 KB)

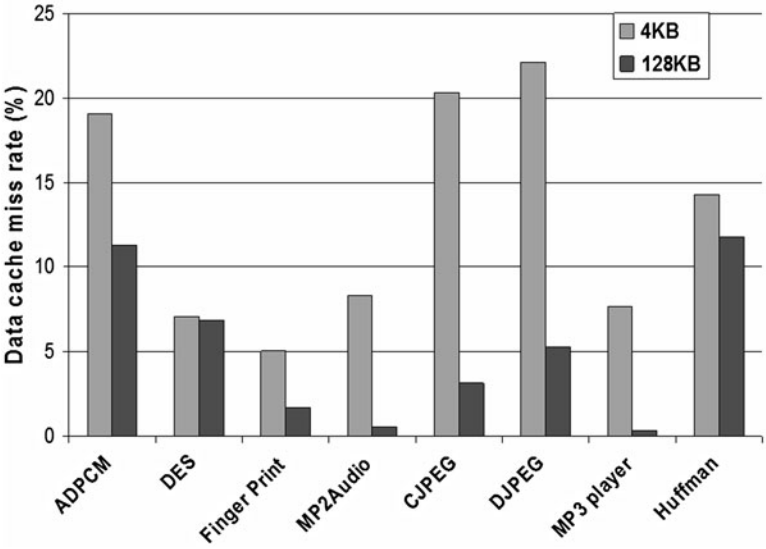


Fig. 6.9 Data cache miss rate for the set of benchmarks used for performance evaluation

overhead compared to standard encryption. In [15, 16], we also showed that PE-ICE is more efficient than the conventional approach in ensuring data confidentiality and integrity. The conventional approach is called generic composition and consists in

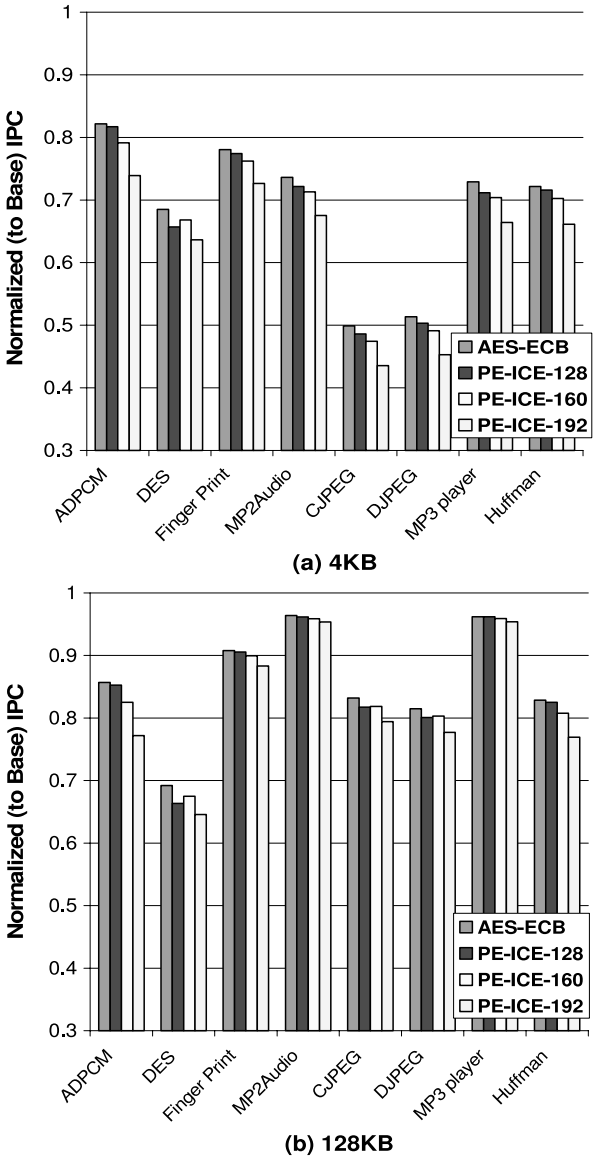


Fig. 6.10 Run time overhead of AES-ECB encryption and of PE-ICE configurations for two data cache sizes (4 KB–128 KB)

chaining encryption with authentication performed with a message authentication code algorithm. We showed that a generic composition scheme can require 50% more hardware resources than PE-ICE and has an 18% overhead compared to an encryption only scheme. In order to decrease the on chip memory overhead and be sure to prevent replay attacks, we propose to build a tree that uses the block level

AREA as authentication primitive. In the next section, we describes the TEC-Tree (Tamper-Evident Counter Tree) [18].[2]

6.3.3.2 The Tamper-Evident Counter Tree (TEC-Tree)

In the TEC-Tree [18] the authentication primitive f is the Block-level AREA (Fig. 6.4). Thus, the authentication primitive tags its input with a nonce N before ciphering it with a block encryption algorithm in ECB mode and a secret key K kept on-chip. The block level AREA is first applied to the memory blocks to be stored off chip, and then recursively over A-sized groups of nonces used in the last iteration of the recursion. The resulting ciphered blocks are stored in external memory and the nonce used in the ciphering of the last block created—i.e., the root of the TEC-Tree—is kept on chip, making the root tamper resistant. Indeed, without the key, an adversary cannot create a tree node and without the on chip root nonce he cannot replay the tree root. During verification of a data block D, D's branch is brought on-chip and decrypted. The integrity of D is validated if:

- Each decrypted node bears a tag equal to the nonce found in the payload of the node in the tree level immediately above;
- The nonce obtained by decrypting the highest level node matches the on chip nonce.

The tree update procedure consists in:

- Loading D's branch decrypting nodes,
- Updating nonces,
- Re-encrypting nodes.

TEC-Tree authentication and update procedures are both parallelizable because f operates on independently generated inputs: the nonces. The distinctive characteristic of TEC-Tree is that it allows for data confidentiality. Indeed, as its authentication primitive is based on a block encryption function, the application of this primitive on the leaf nodes (data) encrypts them. The memory overhead[3] MO_{TEC} of TEC-Tree [18] is:

$$MO_{TEC} = \frac{2}{A-1}.$$

[2]TEC-Tree uses nonce in its design as redundancy for the block level AREA techniques. In [15], we first proposed to build a tree—called PRV-Tree, for PE-ICE protected Random Value Tree—similar to TEC-Tree except that it uses random numbers instead of nonces. The purpose of the PRV-Tree is to decrease the probability for an adversary of succeeding a replay by increasing the length of the random number while limiting the on chip memory overhead to the storage of a single random number (the root of PRV-Tree).

[3][18] give a different formula for their memory overhead because they consider ways to optimize it (e.g. the use of the address in the constitution of the nonce). For the sake of clarity, we give a simplified formula of the TEC-Tree memory overhead by considering that the whole nonce is made of a counter value.

Table 6.1 Architectural Parameters for Simulation

	Merkle Tree	PAT (Parallelizable Authentication Tree)	TEC-Tree (Tamper-Evident Counter Tree)
Splicing, Spoofing Replay resistance	Yes	Yes	Yes
Parallelizability	Tree Authentication only	Tree Authentication and Update	Tree Authentication and Update
Data Confidentiality	No	No	Yes
Memory Overhead	$1/(A-1)$	$3/2(A-1)$	$2/(A-1)$

Comparison with Existing Trees Table 6.1 sums up the properties of the existing integrity trees. PAT and TEC-Tree are both parallelizable for tree authentication and update procedures while preventing all the attacks described in the state of the art. TEC-Tree additionally provides data confidentiality. However, TEC-Tree and PAT also have a higher off chip memory overhead than Merkle Tree, in particular because they require storage of additional meta-data, the nonces.

6.3.3.3 AES-TASC

Memory Security Architecture The AES-TASC (Time Address Segment Cipher) approach, shown in Fig. 6.11, relies on a hardware security core (HSC) fashioned from FPGA logic and embedded memory that is able to manage different security levels according to the data address received from the processor. A small lookup table (the security memory map or SMM) is included in the core to store the security level of memory segments accessed by tasks. Three security levels are possible for each memory segment: confidentiality only, confidentiality and integrity, or no security. The implementation of the security policy in the SMM is independent of the processor and associated operating system. The configuration of the SMM and the rest of the core is contained in the encrypted FPGA bitstream. The isolation of the SMM makes it secure against software modifications at the expense of software level flexibility. New multi-task applications require a new FPGA bitstream to achieve a new memory security protocol.

Security Level Management The increased use of soft and hard-core processors in FPGAs has facilitated the use of operating systems in FPGA based systems. The use of an OS provides a natural partitioning of the application code and data. In Fig. 6.11, the application instructions and stack data of Task 1 have different security levels. In this case, the application designer may wish to keep task processor data secure to prevent copying. The application code may be less sensitive, consequently eliminating the need for security. Our approach is designed to be used in conjunction with a memory management unit (MMU). This unit ensures that a task will not read or write memory segments that are not associated with it, which

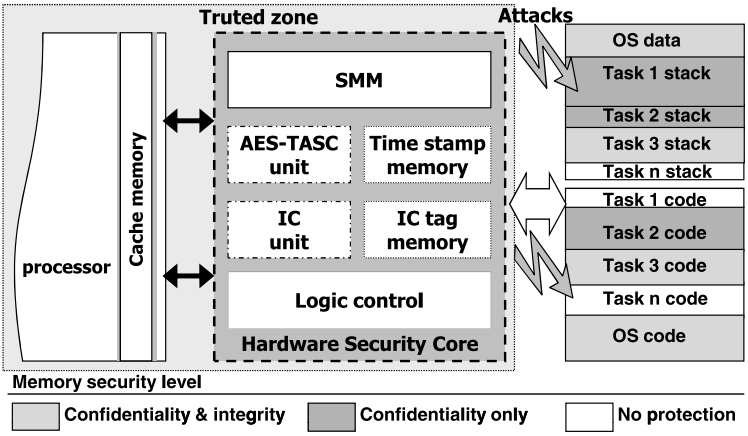


Fig. 6.11 Overview of the memory security system

creates a security risk if the security levels differ. The availability of configurable security levels has an advantage over requiring all memory to perform at the highest security level of confidentiality and integrity checking. The amount of on chip memory required to store tags for integrity checking can be reduced if only external memory that requires security is protected. In addition, the latency and dynamic power of unprotected memory accesses is minimized since unneeded security processing is avoided. FPGA reconfigurability enables optimization of the required on chip storage and modification via a new bitstream.

Memory Security Core Architecture Confidentiality in our system is similar to the AES-based encryption scheme called Binary Additive Stream Cipher [29]. Rather than encrypting write data directly, our approach first generates a *keystream* using AES that operates using a secret key stored in the FPGA bitstream. In our implementation, a time stamp value, the data address, and the segment ID of the write data are used as input to an AES encryption circuit to generate the keystream. These parameters are required to protect the system against spoofing, replay and reallocation attacks. This keystream is then XORed with the data to generate ciphertext that can be transferred outside the FPGA. The time stamp is incremented during each cache line write. The same segment ID is used for all cache lines belonging to a particular application segment (i.e. same level of protection). The advantage of the AES-TASC (AES in time address segment counter mode) approach over direct data encryption of the write data can be seen during data reads. Keystream generation can start immediately after the read address is known for read accesses. After the data is retrieved, a simple, fast XOR operation is needed to recover the plaintext. If direct data encryption is used, the decryption process would require many FPGA cycles after the encrypted data arrives at the FPGA. Thus, the use of AES-TASC significantly reduces the read latency of security. One limitation of this approach is the need to store the time stamp (TS) values for each data value (usually a cache

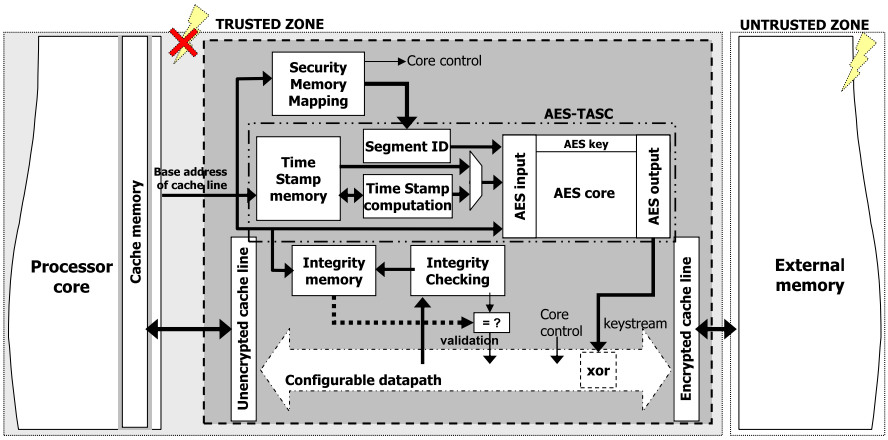


Fig. 6.12 Hardware Security Core Architecture

line) on chip so it can be used later to check data reads. A high level view of the placement of security blocks is given in Fig. 6.12.

The percentage performance loss due to our security scheme is higher for systems that include smaller caches. This is to be expected, since smaller caches are likely to have a larger number of memory accesses, increasing the average fetch latency. Performances are directly related to the designers security policy. A very conservative approach in terms of security will lead to a larger performance penalty whereas a fine tune security policy will lead to a limited reduction in performance. In practice, the use of programmable protection allows the impact on application performance to be reduced compared to uniform protection. An average of 12% performance reduction was observed for a set of applications from multimedia and communication domains. This result compares favorably with other cryptographic approaches where up to 50% performance loss can be observed. The same remarks apply to the memory overhead, which is directly impacted by time stamp and integrity tag values that consume secured on chip embedded memory and energy efficiency. Experiments that have been conducted have shown the benefit of a flexible approach to security.

6.4 Secure Bitstream Management

FPGAs are very specific silicon devices. Like microprocessors, they can be programmed and reprogrammed, but in the case of programmable logic devices the architecture of the chip is changed according to a binary file called a bitstream. In contrast, microprocessors are only programmed with instructions. If this hardware reconfiguration capability is attractive for low-volume applications and opens a wide range of opportunities for engineers or researchers, it can be a drawback in the field of security sensitive applications. Therefore in the following section, we analyze the impact of this feature on the robustness of a secure system.

6.4.1 Threat Model

The first threat to a bitstream is an attacker who succeeds in retrieving it from the system. The easiest way is to use the read-back capabilities of most FPGAs. This function, generally used for debugging, allows the bitstream to be extracted from an FPGA device at run time. Of course, this feature has to be disabled in a secure context, but this not sufficient. For low cost SRAM FPGAs, bitstream retrieval is very simple, even without a read-back mechanism. Indeed, the bitstream is stored in an external non-volatile memory, so the attacker can probe the data line between the FPGA device and this memory in order to access the bitstream. This threat does not exist for non-volatile FPGAs because configuration data are stored inside the device, so only intrusive attacks are possible. Bitstream encryption mechanisms are available for most advanced FPGAs. A secret encryption key is stored inside the programmable device, while for volatile FPGAs, an external battery is used to store key values. Thus the device accepts an encrypted bitstream and uses its dedicated decryption engine to obtain deciphered data. Attackers cannot decrypt the bitstream without the secret key. With this feature, attackers have to discover the secret key using intrusive attacks to recover bitstream data, for example. In FPGA devices with embedded configuration memory, if the designer has taken the trouble to prevent read-back, bitstream retrieval from the device is only possible using invasive attacks. However, if the design is intended to be updated during its lifetime, the bitstream will travel through insecure channels such as public networks, and the bitstream consequently needs to be protected using cryptographic mechanisms even for non-volatile FPGAs.

When an attacker retrieves a plaintext FPGA bitstream, many data are accessible and may be critical. The first threat is bitstream reverse engineering, some software projects claim to succeed in Xilinx bitstream reverse engineering [32]. Thus attackers could access all data stored in the FPGA architecture, possibly secret cryptographic keys or non-public cryptographic algorithms. Attackers can also inject their own bitstream into the programmable device thereby rendering reverse engineering unnecessary, instead they can simply create a dummy system. For instance if the FPGA chip is used to encrypt data written to a hard disk drive, the attacker could build a system that does not cipher data. Another threat is fault injection into the bitstream [7]. Small modifications can deeply modify the system without the knowledge of the architecture. The last FPGA drawback is cloning. An attacker in possession of the configuration data can *clone* the device ignoring potential intellectual property rights. This is not a real security weakness, but rather an industrial threat that is specific to programmable devices. A bitstream is the image of the FPGA underlying architecture, it is similar to the ASIC "layout", for that reason, configuration data have to be well protected. In the following section, we provide a more detailed state of the art on these aspects.

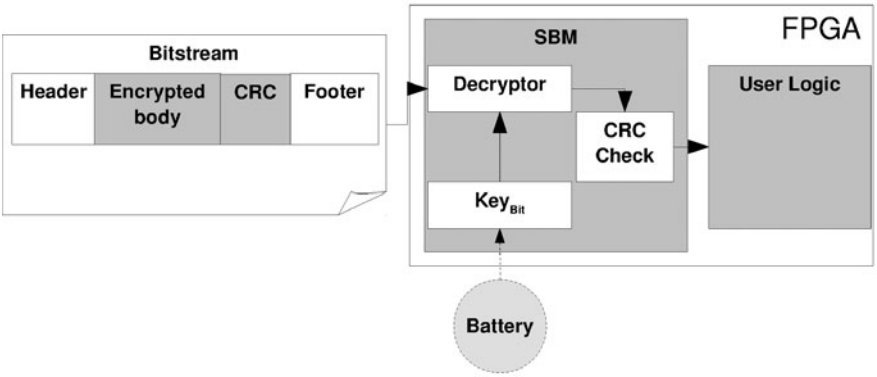


Fig. 6.13 Overview of the static logic needed to enable bitstream encryption

6.4.2 State of the Art

FPGA vendors are generally sensitive to the security issues of FPGA based designs; hence they provide tools and mechanisms that allow system designers to ensure an acceptable level of security according to their requirements. Below we review currently available security features in main FPGA devices. However, we only consider the leading FPGA vendors (Xilinx, Altera, Lattice and Actel).

6.4.2.1 Bitstream Encryption

Most often, the first security feature offered by FPGA vendors is bitstream encryption. This is very attractive even for applications that have no security concerns. Bitstream encryption was originally proposed to protect the confidentiality of any intellectual property included in the design. Without this precaution anyone gaining access to the bitstream can at least *clone* the design in another FPGA.

To allow system designers to encrypt bitstream and thus the FPGA chip to decrypt it, FPGA vendors must add to their devices a decryption engine, a non-volatile key register and control logic that manages configuration (see Fig. 6.13). During the manufacturing stage of a product, the system designer introduces a random and secret key in the non-volatile memory. Then each time the FPGA device reloads its configuration, the decryption engine decrypts the bitstream that comes from the configuration port and transmits the result to the configuration logic. Thanks to this mechanism, bitstream confidentiality is ensured. Today many FPGA devices include a decryption engine to protect the intellectual property of their customers. Currently, Xilinx and Altera provide encrypted bitstream solutions for high end FPGA chips in the Virtex and Stratix families respectively [8, 43]. On the other hand, Lattice and Actel provide solutions even for low cost devices [6, 27]. The latest Xilinx low cost devices also include bitstream encryption but only for large FPGA matrices

(Spartan 6, Virtex 6). Since volatile FPGA vendors generally choose to avoid expensive Flash process, the FPGAs require an additional external battery to store the bitstream decryption key value. However, the latest Xilinx devices include fuses that allow the user to store the cryptographic key without an external battery. Bitstream encryption is a non-negligible cost for FPGA vendors; they need to add a decryption engine that provides a reasonable throughput to be sure the configuration time is acceptable. However for Flash-based FPGAs such as Actel, the configuration time is less critical since they do not need to decrypt the bitstream at each power up. The static logic includes the decryption engine and the key memory, so the security level of the bitstream protection is determined by the FPGA vendor. For instance, if the FPGA vendor does not take side channel attacks on configuration logic into account, a lot of mechanisms that can be built on bitstream trustworthiness will be useless.

6.4.2.2 Bitstream Integrity Checking

If a message is only encrypted, nothing attests to its integrity, because classical encryption does not ensure it. Therefore, in addition to encryption, FPGA vendors provide mechanisms to check bitstream integrity at each configuration. In the absence of any integrity checking mechanisms, attackers can easily modify an encrypted bitstream. However, since they do not know the decryption key value, they cannot predict the effect of the modification. Two main effects can be obtained: (i) an incorrect bitstream loaded in the FPGA chip could damage the chip and the whole FPGA based system; (ii) the bitstream does not damage the FPGA device but modifies the behavior of a part or of the whole design in an uncontrollable way. In the latter case, attackers can target a particular area of the bitstream (and therefore of the design), for instance they could tamper with the embedded RNG to generate weak keys. So FPGA vendors have to include mechanisms that are able to detect bitstream modification. This feature has to be provided by FPGA manufacturers, not by system designers, at least during the bitstream loading time. Most FPGAs use 32- or 16-bit cyclic redundancy code (CRC) combined with AES-CBC encryption. However, the primary aim of CRC is to detect and correct errors during bitstream transmission. As this is not a cryptographically secure mechanism, the probability of loading an unauthentic bitstream is not negligible, even if the bitstream is encrypted. In the FPGA context, a corrupted bitstream can destroy the device by causing short cuts. This threat can be considered as a fault injected into the configuration logic. The latest Actel devices [6] and series 6 Xilinx FPGAs (Spartan and Virtex 6) include cryptographically secure mechanisms to ensure integrity. Actel claims to implement a real cryptographic integrity checking mechanism by using an AES block for both decryption (CBC mode) and integrity checking (AES-based MAC), a message authentication code is included in the bitstream and the configuration logic checks it before starting the design (Fig. 6.14). For Actel, configuration time is not critical since the bitstream is loaded only once. Series 6 Xilinx FPGAs [44] include an integrity checking engine in the static logic using the SHA-1 algorithm. This engine checks bitstream trustworthiness while the AES engine is decrypting it, in order to

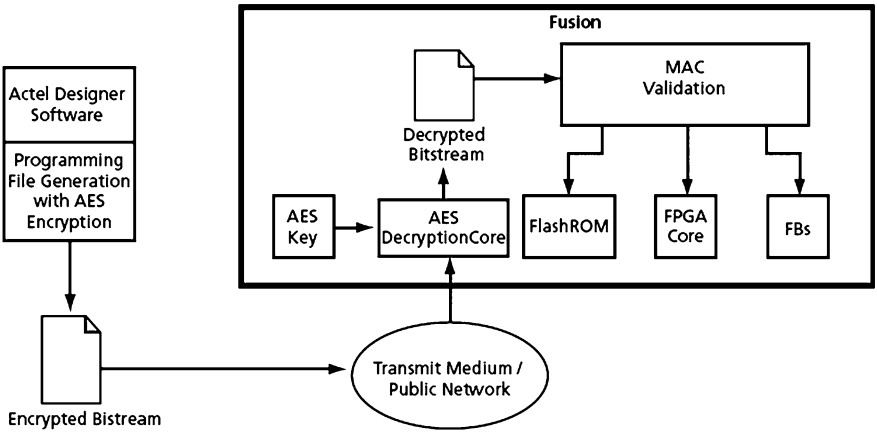


Fig. 6.14 Actel's integrity bitstream mechanism

save configuration time. This mechanism is based on an HMAC algorithm that uses SHA cryptosystem with a 256-bit key.

Cryptographic mechanisms that ensure both integrity and confidentiality already exist, they are known as Authenticated Encryption (AE) algorithms. Therefore, the academic literature proposes secure schemes that are suitable even for volatile FPGAs. These solutions can be implemented using a block cipher in a particular mode of operation, [35] proposes to use the EAX mode and provides time and area overhead which appear to be suitable for volatile FPGAs. Similarly, [12] proposes to use two AES cores and [26] suggests the AES-GCM mode.

6.4.2.3 Locking Reprogramming

In most SRAM FPGAs, there are no non-volatile elements and so the bitstream protection key memory is powered by an external battery. The corollary is that an attacker with physical access to the system can remove the battery and erase the key. Moreover, even when the key is initialized and the battery present, the FPGA chip will accept unencrypted bitstream that an attacker can easily generate. The effect on security is that attacker can load a malicious bitstream. One solution is to add a design authentication key in the bitstream used by an external trusted party to authenticate the design. The other solution, which is more convenient since it does not require an external party, is to lock the FPGA device to only accept encrypted bitstream. Another extreme solution is to prevent any further configuration.

Actel has provided this locking feature for a long time, obviously in anti-fuse based FPGAs since these are programmable only once, but also in the ProASIC 3 and Fusion families [4]. This mechanism, called Flash Lock, acts like a password, the system designer sets the key (password) in the FPGA chip then the SD can choose among different security parameters. The designer can prevent the FPGA

bitstream being read or written without the proper password, and once unlocked, the bitstream can be sent in plaintext or only encrypted depending on SD policy. If the SD wants to allow further remote updating without revealing the password key, he can configure the FPGA device to accept only encrypted bitstream, in which case the bitstream can be sent over an untrusted network; in this case FPGA chip is not locked (i.e. it accepts encrypted bitstreams without the password).

A more drastic solution provided by Actel in their ProASIC and Fusion families is to disable any further reprogramming even with a password key. Obviously any further remote update is then impossible. However this solution can be interesting for very sensitive applications that need to avoid cryptography usage when possible, mainly because secret keys can be retrieved in many different ways (such as physical attacks or even social engineering).

Finally, the latest Xilinx and Altera FPGAs include a new feature that enables system designers to lock the reprogramming of the device. To avoid the cost of Flash technology, Xilinx uses eFuse technology to implement this feature. The solution is flexible since a battery powered key can still be used depending on application requirements, if fast key erasure is needed, battery solutions are best. In order to use eFuse memory, the system designer has first to program a key in this memory. At this point he can still read and write the bitstream key in order to check the value, and if the value is valid, can lock the FPGA device by programming another eFuse register that disables any further read and write access to the key register; moreover one bit in this control register allows the SD to constrain the FPGA chip to only load bitstreams that are encrypted with the eFuse key. In this way, attackers who do not know the bitstream key cannot load a malicious design in the FPGA.

6.4.3 Remote Reconfiguration

Remote updating of hardware systems is a convenient service enabled by FPGA-based systems. This service is essential in applications like space-based FPGA systems or set top boxes. However, remote update schemes have to consider replay attacks, as described in Fig. 6.15. This attack consists in simply recording a bitstream corresponding to a version of the FPGA design, and then replaying it later to reintroduce security breaches that have been corrected by the system designer in the latest bitstream version. According to current bitstream encryption mechanisms, the attack is possible even if the bitstream is encrypted.

6.4.3.1 Related Work

Few academic or industrial studies have addressed this attack. In [35] and [12] the keyed hash is computed only over the received bitstream. As previously mentioned, most current FPGA vendors do not provide strong bitstream integrity checking mechanisms, and none have addressed the replay attack issue. In all cases, the

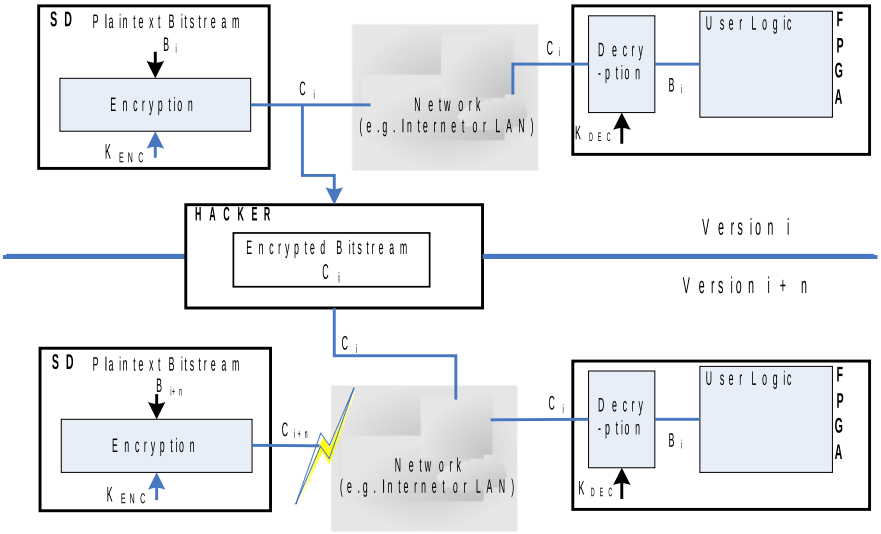


Fig. 6.15 Replay attack of an FPGA bitstream

FPGA configuration logic is unable to distinguish between different (keyed hash, bitstream) pairs legitimately generated by the SD in the past. As a result, an adversary who replays a bitstream and its keyed hash will succeed in his attacks. A replay is particularly dangerous for system security because even if bitstream encryption is enabled by the FPGA's static logic, it allows for system downgrading. The purpose of a bitstream update triggered by the SD may be to remove system vulnerabilities, so by replaying the previous FPGA configuration, an attacker can effectively preclude security-critical updates. In the following sections, we provide solutions to counter such an attack.

Existing bitstream integrity solutions prevent spoofing of the bitstream but are unable to prevent a replay attack and the bitstream is consequently exposed to system downgrade threats. In [6, 12, 35] the keyed hash is computed only over the received bitstream. For that reason, the FPGA configuration logic is unable to distinguish between different (keyed hash, bitstream) pairs legitimately generated by the SD in the past. As a result, an adversary who replays a bitstream and its keyed hash will succeed in his attack. Recent work on reconfigurable trusted computing proposes FPGA-based implementations of the Trusted Platform Module (TPM) [14]. This work leverages TPM functionalities to provide a secure update of the FPGA bitstream. In addition, [36] proposes an implementation of TPM on current FPGA technologies that does not require bitstream encryption. [36] assumes that reverse engineering of the bitstream is too difficult to achieve and relies on a trusted external non-volatile memory.

[13] introduces the issue of bitstream replay attacks. The author suggests two different avenues of research to solve the problem. The first requires the SD to implement additional security features in the user logic to send authenticated messages

to a trusted authority, who then attests to the version of the running FPGA configuration. This suggestion is explored in this section (see 6.4.3.2). While this approach fits the general reasoning of FPGA vendors, i.e., that an SD who wants a specific functionality should pay for it himself by developing it in the user logic rather than it being hardwired and supplied to everyone who buys FPGAs, we do not believe it is the best solution. Firstly, it requires the implementation of a cryptographic engine in the user logic to set up an authentication channel and a challenge-response protocol with the SD. This is not an efficient solution, since the one already provided in secure FPGAs for bitstream encryption could also be used for that purpose if the scheme was implemented in the static logic (i.e. as part of the bitstream loading logic). Secondly, the SD in need of the replay-resistant feature for his design is not necessarily a security expert; hence, custom implementation of a replay-resistant system can result in unreliable solutions. Consequently, we suggest mechanisms that require less security expertise and allow the SD to lock the FPGA based system to a particular bitstream version.

[13] also suggests a second approach that use nonces in the authentication process to ensure the freshness of the bitstream. [13] does not, however, define the architecture and protocols that would be necessary to build a replay-resistant bitstream authentication mechanism. As this solution is developed and evaluated in [9], we do not present it here and interested readers should refer to this paper for further information.

Based on these studies, it is clear that system designers lack efficient solutions to prevent bitstream downgrade and more generally to ensure bitstream security. Therefore, in the following sections we propose two possible solutions along with their advantages, drawbacks and limitations.

6.4.3.2 Contribution to Remote Configuration Security (1): FPGA Polling

Principle The first solution can be applied to any FPGA device that supports bitstream encryption and integrity checking. Bitstream encryption is used to hide a value related to the bitstream version; each new release of the bitstream will contain a new value, called TAG. In addition to this TAG, the encrypted bitstream contains a unique value that aims at identifying a particular device, each device contains its own identifier, called K_{ID}. Figure 6.16 describes this layout.

For security reasons, inner FPGA logic (user logic) cannot access the bitstream encryption key. Thus designers cannot use this key directly to perform cryptographic operations. However, designers can hide secrets in the encrypted configuration bitstream. These values may be secret or private keys stored in user logic such as look up tables or embedded RAM blocks.

As suggested by Saar Drimer in [12] these secrets can be used to authenticate the bitstream. If one bitstream decryption key is kept secret by the system designer for each FPGA chip, only the FPGA chip containing the relevant key can decrypt the authentication secret. This allows authentication of the design running on the FPGA. The key can also be used to check if the version of the secret corresponds to

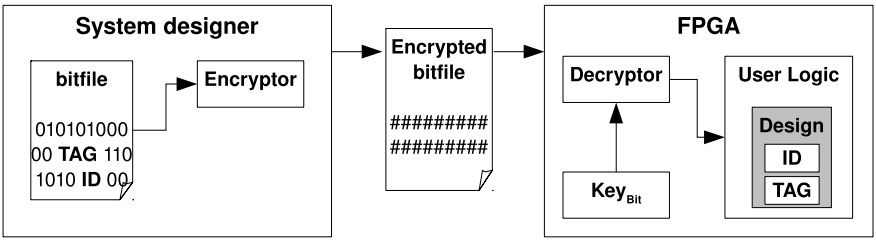


Fig. 6.16 Key equipment needed for Solution 1

the current genuine version of the design. These two values are used in the proposed protocol to ensure authentication of both the transaction and the current bitstream version. Since this solution can be applied in any FPGA chip that includes a bitstream encryption feature, we assume that like in most SRAM-based FPGAs, there is no internal non-volatile memory and that the FPGA design cannot store the current version number because it does not have an embedded trusted reference. In this solution, the SD needs to implement a cryptographic engine and glue logic able to perform authentication with an external trusted party (TP) to authenticate the bitstream version. The proposed solution consists in regular polling of the FPGA bitstream version by the external trusted party who also knows the TAG and the K_{ID} values. The trusted party could be the system designer himself but it could also be an external processor or system. The trusted party regularly sends a nonce to the FPGA chip that replies with an encrypted value. This encrypted value is the concatenation of the nonce with the TAG ciphered with K_{ID}. The trusted party can check that the bitstream version is valid by checking the TAG value, but also that the response is not a replay using the nonce. These two checks are made using K_{ID} to decipher the response; they are thus authenticated according to the FPGA device concerned.

In the following, we consider that the FPGA device and the trusted party are located on the same board, that all the considerations of this solution can be applied to a remote trusted party (access through Internet for instance) and since all the communications are tagged, authenticated and encrypted, that they are not subject to attacks. How the SD securely updates the trusted party is beyond the scope of this book, although he could a secure microprocessor able to securely store key materials, for instance, but in this case, would need to include a mechanism in the device to avoid replay attacks on trusted party updates.

Platform Initialization First, the SD initializes the platform by setting a secret key to decrypt the bitstream. To do so, he uses the appropriate tools and mechanisms provided by FPGA vendors. This key must remain secret. Next, he initializes the trusted party that is located on the FPGA based device, loads a TAG that does not need to be secret (0 for instance) and a key K_{ID} that will be used to ensure platform authentication. He can optionally load the initial bitstream in the FPGA configuration memory (which could be inside the FPGA chip in the case of non-volatile FPGAs).

Remote Update Process Detail Once the platform is initialized and the bitstream is loaded in the FPGA user logic, polling of the trusted party (TP) can begin. The goal of polling is to check that the bitstream version is genuine.

The process described here requires that an encryption engine is embedded in the FPGA device that will be used to authenticate the current version of the bitstream used. The encryption algorithm can be of any kind (i.e. stream cipher, block cipher, asymmetric) but a symmetric block cipher may be best because these algorithms are compact and easy to implement in FPGAs. A standard AES engine can be used for this purpose.

When the SD wants to provide a new version of his FPGA design, he has to remotely modify the FPGA bitstream and also the trusted party TAG value. However, he can send the bitstream securely since its integrity and confidentiality are ensured by the FPGA device. Alternatively, he can send the new TAG version to the TP using a secure network connection (for instance SSL). Once the bitstream is loaded into the FPGA chip and the TAG is changed in the TP, the polling process can (re)start.

Assuming that the AES algorithm is used, the proposed polling mechanism can be described as follows:

1. The trusted party sends a nonce to the FPGA device. This nonce will be used later to check that the FPGA response is genuine, i.e. not a replay of an old FPGA response. The method used to send the nonce is platform-dependent. For instance it could be a serial link between the TP and the FPGA, or if the TP is a remote server, it could be over Internet.
2. The FPGA chip receives the nonce and computes its response. To do so, first it concatenates the nonce and its current TAG value. Then it authenticates this concatenation using its embedded signature engine and its identity key. Since the K_{ID} value is only known by the SD, the FPGA device and the TP, this response cannot be generated by an attacker.
3. Once the answer is computed, the FPGA chip sends the value to the TP, and since the response is encrypted, an attacker has almost no chance of generating a valid response. If an attacker tries to modify this answer, the TP will reject it, which can only result in a denial of service.
4. On reception, the TP decrypts the FPGA response using K_{ID}. The decrypted value is then compared to the concatenation of its own copy of the nonce and its TAG value.
   a. If these values are correct, TP can continue its polling process as the current bitstream version is genuine.
   b. If only the TAG value is different, this means that the FPGA device is running a different bitstream version than the genuine one. TP then applies the system designer policy, which could be to turn off the whole system, or to reload a genuine version in the FPGA configuration memory.
   c. If only the nonce is different, this means an attacker is probably trying to replay an old answer given by the FPGA chip as the genuine version. To do this, he lets the FPGA device be upgraded to the genuine version, then records a set of FPGA response. After this stage, the attacker performs a downgrade of the FPGA bitstream (recorded from a previous remote update), and then

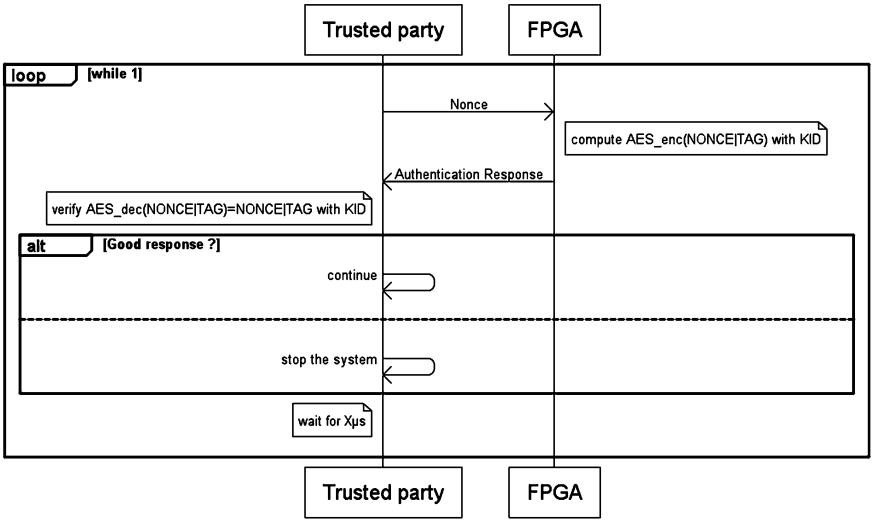


Fig. 6.17 Graphical representation of the protocol

tries to replay the old FPGA answers. The nonce ensures that this attack is not applicable. Once again the TP applies the system designer policy.

d. If both the TAG and the nonce are invalid, the cause of the error is harder to determine, maybe it is simply a transmission error due to the channel between the FPGA chip and the TP. So depending on the SD policy, the trusted party may try to poll the FPGA device one more time discarding this failure.

Figure 6.17 is a simplified graphic representation of the polling procedure.

Security Analysis The main security drawback of this solution is that the K_{ID} value has to be shared between three different entities: the SD, the FPGA chip and the trusted party. This means that attacks are more likely and that more entities have to be trustworthy. Next, the system designer has to implement a decryption engine in the user logic that is subject to physical attacks like any cryptographic algorithm implementation. However, the SD may not be a security expert and could consequently introduce weaknesses in the protocol or in the implementation. The update of the trusted entity must also be realized by the SD and can result in threats in his protocol. The polling frequency has also an impact on system security. Security does not need to be very high to counter an attack between two polls. The attacker first needs to replay an old version. This operation can take a relatively long time (a few milliseconds) depending on the FPGA bitstream loading speed. Next the attacker has to perform the attack. Finally he has to reload the genuine bitstream before the next polling of the TP. Therefore the SD should choose the pulling frequency according to requirements of his particular threat model.

Cost Evaluation The authentication engine that generates responses inside the FPGA may be symmetric or asymmetric. Both allow secure implementation of the

solution, but the symmetric solution is less convenient than the asymmetric one. With a symmetric cipher, if anyone needs to check the key, they need to have this key. The solution can be made with an AES block used as a MAC. Whereas an asymmetric cipher can be checked without compromising its secrecy. In counterpart, asymmetric ciphers are more compute intensive operations. So the choice will depend on the application.

It should be noted that an existing cryptographic engine in the user logic can be reused by the SD to perform authentication responses. The mean performance of this engine will be reduced since some operations will be reserved for the version checking protocol, meaning the application cannot use it at this time. The overhead is directly related to the polling frequency specified by the trusted party.

Assuming that the SD decides to use a dedicated cryptographic engine to implement this solution, the cost could be relatively high; especially if the FPGA device of the application has limited resources. Obviously the asymmetric approach requires much more logic and time to compute the response. But the cost of the symmetric approach is also non-negligible if the AES engine is not reused by the SD. The cost of the additional logic gates needed to implement the protocol is negligible compared to the cost of a hardware cryptographic engine.

Conclusion The proposed solution allows the TP to securely monitor the FPGA bitstream version using existing SRAM based devices. However this is not the ideal, perfectly convenient solution. First it entails a non-negligible cost for the SD. Second, the entity that wants to check the design version has to question the device. The SD must also find a polling frequency that is not too high in order to maintain reasonable performance, and not too low to avoid replay attacks between two polling sequences. Moreover, this process forces the SD to include secure key management in his design. He must manage the bitstream key, embed an identity key and an update TAG inside the encrypted bitstream. All these drawbacks are due to the fact that the FPGA device itself does not check if a bitstream replay attack is underway. Unfortunately, this is the only way to ensure update security for most current FPGAs. The ideal solution would be to lock the FPGA chip to only one bitstream version, and provide a secure solution to the SD to remotely change the current genuine bitstream version. The following section addresses this need.

6.4.3.3 Contribution to Remote Configuration Security (2): Using Embedded Non-volatile Memory

Finding a solution to lock current FPGAs to a bitstream version is not easy, mainly because it requires that the FPGA is able to store a reference of the current bitstream version TAG. It is possible to implement an AES engine and the comparator, as mentioned in our first proposal above. But dealing with non-volatile storage of the TAG is more problematic. User logic is by nature volatile (even on Flash based FPGAs), and the TAG value cannot be stored inside the bitstream since, in the threat model, the attacker could replay an old bitstream. It cannot be stored in an external memory

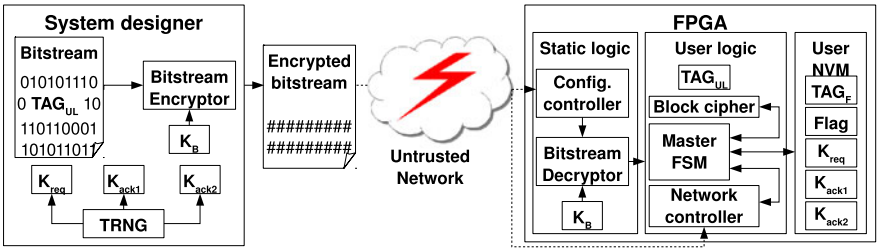


Fig. 6.18 Overview of the equipment needed for the second proposal

either since the attacker could replay the memory content. So the TAG value must be stored in an embedded non-volatile memory. This memory must be protected from external dumping and tampering, and must also be usable by user logic to enable the secure TAG update that will be performed by the FPGA design. Some Actel FPGAs [6] include a feature called Flash ROM. This non-volatile memory can be read from inner logic and can be programmed via JTAG with an encrypted bitstream. However the FROM is not writable from inner logic, so a secure TAG update mechanism cannot be used. In fact, the ideal solution would be to enable the FPGA device to perform the TAG update by itself, although such a mechanism could not use external JTAG since an attacker would be able to replay any communication performed using JTAG. For that reason, a FROM could not be used to enhance solution 1. However, FUSION FPGAs from Actel [5] include a more interesting feature: a user flash memory. This non-volatile memory is accessible from user logic for both read and write operations. In addition, using a secret enables it to be protected against dumping and tampering from external inputs and outputs such as the JTAG port. Thanks to this feature, solutions can be found to lock such an FPGA chip to a particular bitstream version and to remotely modify the version number.

Next we present a minimum but nevertheless secure solution, whose goal is to minimize hardware overhead and key management for the system designer. This is why the solution does not include complex cryptographic algorithms.

Principle The goal of this secure update mechanism is to lock the FPGA to a specific version in order to prevent replay attacks.

Generic Design Overview Figure 6.18 shows the FPGA design that enables a secure remote update of bitstream mechanism to be implemented. The FPGA is composed of three parts. The first part, static logic, is hard-wired and cannot be configured. It contains a deciphering module that protects the confidentiality and integrity of the bitstream, and its key. This key, named K, is only known by the FPGA and the SD. The second part, user logic, can be configured by the SD and contains a bitstream version verification mechanism. It is composed of a finite state machine (FSM) able to manage:

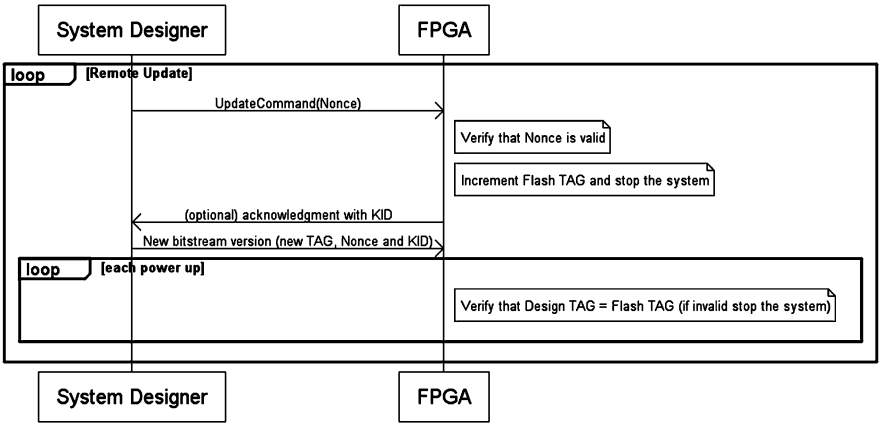


Fig. 6.19 Graphical representation of the protocol

- A network controller to enable the update to be performed remotely.
- A non-volatile memory controller to store a first power up flag, the current bitstream version number and keys shared with the SD.
- A block cipher to ensure mutual authentication of the FPGA and the SD.

The third part, user NVM, contains three keys shared with the SD. They must be unique for each FPGA and are used to encrypt the tag:

- K_{req}: for the Update command.
- K_{ack1}: for the Update command acknowledgment.
- K_{ack2}: for the new bitstream version and acknowledgment of start up on the correct device.

Since the goal is to lock the FPGA to a particular version, the NVM also contains the value indicating the current genuine version. This value, named TAG, can be only incremented by the SD. It will be compared to the tag contained in the user logic, also named TAG_{UL} (refer to Fig. 6.18). Each bitstream version contains its own TAG_{UL}. In practice, it is a constant in the design source code: version zero is tagged with a zero, version one with a one, and so on. The NVM is written the first time from outside FPGA chip in a trusted zone before being locked using the FPGA vendor mechanism [3]. After locking, the NVM can be read and written only from the user logic.

Update Process Figure 6.19 focuses on communications between the SD and the FPGA. It explains the process used to check that the current bitstream version is genuine and to securely implement this non-volatile value for a future update. The update process is described in more detail below.

Update command: This command increments TAG_F to prepare the FPGA to an update. The SD sends the update command containing the tag encrypted with the K_{req} as cipher key to the FPGA. After decryption, the FPGA compares the tag

contained in his own bitstream (TAG_{UL}) and the tag sent by the presumed SD. If they are different, the FPGA continues to work and waits for a new update command. Otherwise it implements the TAG_F and starts to encrypt the new tag with the cipher key K_{ack1}. To inform the SD that the tag increment command has been received, the FPGA sends the result of the encryption. The design is stopped.

Download a new bitstream version: The SD sends the new ciphered bitstream, with its MAC, to the FPGA. When the new design starts up, the FPGA performs the new tag encryption using K_{ack2} as cipher key and sends the result to the SD. This acknowledgment informs the SD that the new bitstream has been correctly downloaded to the right FPGA and that the design has started.

Bitstream version verification: Each time the design starts up, it checks itself that TAG_{UL} and TAG_F are the same. If they are different, a replay attack has been detected and an alarm (a signal in the design) is triggered that can be used by the SD to apply his policy. He can for instance stop or destroy the system, or enter a degraded mode.

Security Analysis This analysis focuses on bitstream replay and remote DoS attacks. Our scheme assumes that the FPGA vendors encryption and integrity verification mechanisms are secure. For instance, the Actel mechanism implemented in the static logic checks the bitstream integrity using a MAC while the device is still operating. If the MAC validates the bitstream, the device will be erased and programmed. Otherwise, the device will continue to operate uninterrupted and will not take the new bitstream into account. The tag is encrypted with three different keys to prevent replay attacks. Indeed, to avoid the attacker responding by pretending to be the device or the SD, only one-time messages (key-tag pairs) are transmitted over the untrusted network. Since, for a bitstream version, the tag is the same for all the FPGAs, K_{req}, K_{ack1} and K_{ack2} must be unique for each device.

In the step, which is to download a new bitstream version, the new TAG cannot be spoofed because its integrity has been checked thanks to a MAC. For the same reason, an attacker cannot replace this bitstream with his own. The bitstream boot up acknowledgment enables update failures to be detected.

Implementation Considerations The only requirement of this protocol is the presence of an embedded NVM in the FPGA chip and a mechanism that provides bitstream confidentiality and integrity. For instance Spartan3-AN FPGA from Xilinx has an embedded Flash memory. Xilinx also provides a DNA mechanism, but it does not protect bitstream confidentiality. Lattice also provides such services in their XP2 FPGA family. Bitstream confidentiality and integrity are provided, but NVM is used to save a RAM copy and it is less easy to store data.

Demonstration Platform Our demonstration platform is based on an Actel Fusion starter kit (FPGA: Fusion AFS600). It is a non-volatile FPGA with an embedded user flash memory and a confidentiality and integrity mechanism. Figure 6.20 describes this demonstrator. The network is emulated by the RS232 link, the bitstream is downloaded through the JTAG port, and the block cipher is a 3-DES. The fact that

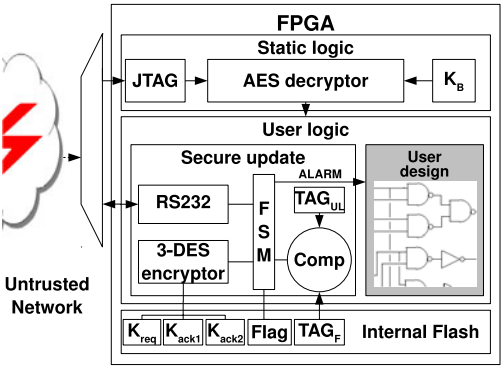


Fig. 6.20 Overview of FPGA design

K_{req}, K_{ack1} and K_{ack2} are stored in the flash memory with TAG_F allows the same bitstream file to be produced for the whole set of FPGAs: i.e. only the three keys that differentiate the device. During the initialization procedure, these three keys and TAG_F are downloaded through the JTAG port before locking.

Figure 6.21 describes the FPGA Master FSM algorithm. In order to reduce the latency, we decided to cipher TAG_{UL} with K_{req} while waiting for the update command. The two ciphered tags are then compared as soon as the SD tag is received. With this improvement, steps 1, 2 and 3 (respectively power up, first power up and authentication) are performed before receiving the SD update command.

Results Table 6.2 summarizes the overhead in terms of clock cycles and time required for each step. The design clocks at 60 MHz. Steps 2 and 3 (respectively first power-up and authentication) are performed during user design execution and do not increase the mechanism performance overhead. Step 4 is not considered here because the performance overhead is not significant compared to FPGA programming (several seconds). Considering all these elements, the performance overhead is estimated at only 54 cycles: step 1 (Power up).

Table 6.3 summarizes the overhead in terms of area. It shows the proportion of FPGA occupied by each component of this secure remote update mechanism implementation. This area overhead can be relativized considering that 3-DES, RS232 and flash memory controller can be reused by the SD. Only the master FSM cannot be reused.

The cost of flash memory is not shown because it is insignificant: 672 bits (including 32 bits for the first power up flag) on the 4 Mbits (0.016%). The SD can use the rest of this physical flash memory for his own purposes.

Conclusion Solution 2 is a communication protocol between the SD and an FPGA platform to update the FPGA configuration while preserving its confidentiality and integrity. This protocol also provides protection against replay attacks and detects update failures. In addition, we show that the corresponding area and performance overheads are negligible, thanks to the reusability of the core.

```
TAG_F : Flash memory tag
TAG_UL : User logic tag
Ek(M) : Encryption of M with K as cipher key\index{Key}
CTAGKx: Tag ciphered by K_x
```

Step 1: Power-up

```
1  Read (TAG_F)
2  if (TAG_F ≠ TAG_UL) then
3  goto 22
4  end if;
```

Step 2: First power-up

```
5  Read (flag)
6  if (flag = true) then
7  Read (K_ack2)
8  CTAGKack2 := E_Kack2 (TAG_UL)
9  Send(CTAGKack2)
10 end if;
```

Step 3: Authentication

```
11 Read (K_req)
12 CTAGKreq := E_Kreq (TAG_UL)
13 Read (K_ack1)
14 CTAGKack1 := E_Kack1 (TAG_UL)
15 Wait for CMD
16 If (CMD = CTAGKreq) then
```

Step 4: TAG_F incrementation

```
17 Write (TAG_F+1)
18 Send(CTAGKack1)
19 Else
20 goto 15
21 end if;
22 SYSTEM SHUTDOWN
```

Fig. 6.21 Security protocol implementation on FPGA

6.4.3.4 Contribution to Remote Configuration Security (3): Security Architecture for Remote FPGA Update and Monitoring

When no non-volatile embedded memory is available, it is not possible to lock the FPGA to a particular bitstream version without an external trust party. However, one possible solution is that FPGA vendors modify their configuration logic to include

Table 6.2 Performance overhead for the AFS600 device

Step	# Cycles	Duration (μs) F = 60 MHz
1. Power-up	54	0.9
2. First power-up	187	3.1
3. Authentication	175	2.9
4. TAG_F increm.	108	1.8
Total	**524**	**8.7**

Table 6.3 Area overhead for the AFS600 device

Entity	# Tiles	% of Actel AFS600
3-DES	1305	9%
RS232	418	3%
Flash Controller	1005	7%
Master FSM	777	6%
Total	**3505**	**25%**

a version checking mechanism. A low-cost solution for FPGA vendors [9] has been proposed. The version TAG of the FPGA is kept in the configuration logic using a few flash or battery powered SRAM cells. The configuration logic uses the authenticated encryption algorithm CCM to ensure confidentiality, integrity and bitstream freshness. The logical gate cost and configuration time overhead is low regarding current implementation of bitstream encryption and integrity mechanisms. Interested readers should refer to this paper for further information.

6.4.4 FPGA Remote Update: Conclusion

From a security standpoint, remote updating of FPGA based systems is a challenge. We have shown that existing mechanisms aimed at providing bitstream confidentiality and integrity via encryption and authentication fail to prevent bitstream replay and thus system downgrade. Three different solutions were proposed to solve this problem. The first one is applicable to all encrypted FPGAs (volatile and non-volatile). The second requires a secure non-volatile memory that is currently only available in Actel Fusion FPGAs, however we hope that next generation FPGAs will be aware of this threat and will have embedded non-volatile user storage. Finally we proposed a complete solution for FPGA vendors who wish to provide a comprehensive solution to their customers at low cost.

According to Table 6.4, the proposed solutions are unique in the academic and industrial literature, since other solutions fail to take replay attack into account. Our solutions apply to both volatile and non-volatile FPGA devices, by accounting for their particularities. In addition, we propose a new complete solution [9] that pro-

Table 6.4 Secure update solution comparison

Solution	Suitable devices	Cost for FPGA vendors	Development time for system designer	Logic gates cost for system designer	Additional cost
1	All encrypted FPGAs	None	High	High	Regular polling
2	ACTEL Fusion	Low for Flash based FPGAs	Medium	Low	None
3	Currently none	Medium	Low	None	None

vides confidentiality, cryptographic integrity verification, replay attack counter measure and also provides convenient way for the system designer to remotely manage the bitstream update process. All the contributions concerning bitstream security and the study cited in this chapter about FPGA security, are used in the following section to develop a Reconfigurable Cryptographic Platform that benefits from these results. The platform can be considered as a concrete application of the concepts developed throughout this book and, thanks to its context, is close to industrial concerns.

6.5 Example of Board Integration: Toward a Secure Platform

To apply the knowledge described in this book, we use a concrete application. This approach allows us to evaluate the solutions we propose from an industrial point of view. This study was done with the help of a French company called Netheos [31] which develops applications for information security. The company aims to create products dedicated to security, based on FPGAs, to occupy low and middle volume markets. Their main targets are corporate and bank sectors, or even government infrastructures. The platform aims to be configurable to respond to many different needs, and it might be reconfigurable in order to evolve, for instance when a cryptographic algorithm or protocol is broken. The code name of the platform is RCP for Reconfigurable Cryptographic Platform. Objectives with this platform is twofold:

- To increase the security of key management compared with a software only solution. This is done by keeping cryptographic keys inside a piece of hardware where keys are only accessible to perform cryptographic computation, not for reading or writing. These keys are generated by the hardware itself with an embedded true random number generator.
- To achieve high performances for cryptographic algorithms such as RSA or AES schemes. This can be done by having a high bandwidth communication channel with the FPGA chip and by using high-speed hardware accelerators (such as RSA or AES IP cores). Depending on the application, these accelerators can be

instantiated multiple times to achieve even better performances by parallelizing computations.

We intend to use its platform for applications such as giga-bit network encryption, SSL accelerator or off-load engines, VPN accelerators, high-performance Hardware Security Modules, but also for customer specific applications.

6.5.1 Requirements

In the following section we present the features we would like to include in the secure platform.

Scalability The platform needs to be as generic as possible regarding cost, performance and security level. It should be able to support relatively low and high security levels. In the first case, the platform needs to keep costs low and provide reasonable security. In the other cases, performance and security concerns are more important than cost. The more resources included in the FPGA, the more performances can be obtained, for instance by implementing many types of cryptographic algorithm accelerators. The selection of the FPGA model will thus determine the performances that can be achieved with the platform.

To obtain security level scalability of the platform, two types of attacks need to be distinguished:

- **Logical attacks**: that have to be countered by implementing a logically secured hardware and software design.
- **Physical attacks**: that can be prevented by implementing cryptographic algorithms with side channel and fault injection countermeasures, but also by adding a secure surface enclosure such as those proposed by [23]. This type of tamper respondent enclosure, which is widely used for Hardware Security Modules (HSM), will erase the cryptographic key when attacks are detected, thus guaranteeing protection against invasive attacks that are quasi impossible to prevent only using logical mechanisms. With such protection, this platform can hope to achieve FIPS 140-2 certification at level 3, at which key destruction is mandatory under attack.

Target Applications The board targets two applications:

- Cryptographic accelerators: the platform needs at least one high-speed interface to communicate directly over the network or through the host.
- Secure key containers (HSM): One of the most important points for a cryptographic device is secure key management in a logical and physical way. A key must never leave the device without being encrypted, or for some applications, should not leave the device even if it is encrypted.

Regarding the second application, even if confining the key to the secure device makes sense with respect to security, it reduces the usability of the solution. If the

Table 6.5 LX and SX family characteristics

FPGA Family	LXT		SXT	
FPGA Model	XC5VLX30T	XC5VLX50T	XC5VSX35T	XC5VSX50T
Slices	4800	7200	5440	8160
Logic Cells	30720	46080	34816	52224
Block RAM	36	60	84	132
DSP48E slices	32	48	192	288

secure device stops working because an irreversible failure occurs in the hardware, or maybe because an attack is detected, all the cryptographic keys become irreversibly unusable. Therefore most hardware security modules use a master intended to cypher the user keys. Encrypted user keys are stored outside the HSM. A mechanism allows master key backup on a secure medium such as a smart card. For both applications, software and hardware updating should be possible throughout the life cycle of the device, maybe to correct security vulnerabilities or to increase performances. However this feature opens the door to many threats and must be implemented carefully.

6.5.2 Platform Design

FPGA Choice The FPGA device is the most important component and determines the price and the performances of the platform. But it plays also a role in the security of the system. Based on the results of a comparison of current FPGAs, we decided to use a Virtex5 FPGA (in 2008). It provides mechanisms and features that enable the implementation of useful security countermeasures. For instance, it allows for continuous monitoring of the integrity of the loaded bitstream, an embedded thermal sensor can detect dangerous variations in the environment. Bitstream encryption allows cryptographic keys to be hidden in RAM blocks or look up tables. Internal read back of the loaded bitstream can increase the robustness of the integrity check of the configuration. Finally, the partial reconfiguration feature is attractive to implement adaptive and evolutionary design. Since the price of the FPGA chip price represents most of the cost of the board, we designed the platform to support any Virtex5 FPGA that fits an FFG-665 package, so Virtex5 LX30T, LX50T, SX30T and SX50T can be used without PCB modifications. The cheapest FPGA that can be used is Virtex5 LX30T, which has limited DSP and storage capacities. The most attractive FPGA for cryptographic applications is Virtex5 SX50T, because it includes many storage (RAM blocks) and arithmetic elements (DSP blocks) that can be used to implement high performance cryptographic cores (such as RSA, ECC or AES). LX and SX family characteristics are listed in Table 6.5.

Cryptographic Engines One of the main advantages of FPGAs over CPUs is that developers can implement specialized pieces of hardware that allow very efficient

Algorithm	Implementation	Slices	Blocks RAM	Multipliers	Throughput
AES 128 (Encryptor and decryptor. w/HWkey expansion)	Helion Tech	864	0	0	3781 Mb/s
	our	803	5	0	1024 Mb/s
SHA-256	Helion Tech	325	0	0	1722 Mb/s
	our	792	1	0	1144 Mb/s
RSA-1024 (full exponent wo/CRT)	Helion Tech	1800	1	0	20 signature/s
	our	378	2	5	24 signature/s

Fig. 6.22 Comparison of results of the implementation of cryptographic IP cores

cryptographic algorithm implementations. AES hardware implementation is a good example; it is quite easy to implement a dedicated engine that executes an AES round in one clock cycle while keeping a reasonable silicon area. However AES engines are not available on most CPU or SoC platforms.

That is why a set of cryptographic engines was developed. This work focuses on the most widely used cryptographic algorithms such as AES, RSA and SHA-2. Most optimizations were found in academic publications describing algorithmic or architectural enhancement to straightforward algorithm implementations. This work takes advantage of these optimizations and produced efficient and compact implementations. Our main contribution is the comparison of different implementation styles and the development of configurable cryptographic cores that offer large area/throughput/functionalities/security trade offs.

The details of the development of these blocks are not given in this book, but Table (Fig. 6.22) gives a brief comparison of our results using commercially available cores.

Comparing cryptographic engines is not easy [21] mainly because many implementations are available and they do not use the same FPGA family, or do not use the same core functionalities. For this reason, we used the same FPGA family for our comparison (Virtex5) and the IP cores cited have the same functionalities (for instance the AES IP core is able to perform encryption and decryption with hardware key expansion). Finding results in academic publications that allow a fair comparison was not possible, mainly because most of these publications are old and use obsolete FPGA devices (such as Virtex or Virtex2 FPGAs). The results achieved with the IP cores developed during this study are no better than the results achieved by Helion Technology (a private company). However, the goal of the present study was not to design highly efficient IP cores but to develop state of the art cryptographic engines and to use them to build a complete security platform.

Secure Microprocessor Based on the results of the FPGA bitstream security study described in previous sections, we decided to provide the board with a secure microprocessor in addition to the Virtex5 chip. The following arguments justify this choice:

- Since Virtex5 are volatile FPGAs that do not include a non-volatile storage element, it is impossible to store a value inside the chip after power down. Although it is possible to store securely values outside the chip in an encrypted flash memory, a master key has to be used to decrypt this memory, thus, to remain non-volatile, the key must be stored in the FPGA bitstream. However this solution is not applicable when the application requires volume, because each FPGA needs a different bitstream to provide each board with a unique key.
- There is no trusted hardwired TRNG in FPGA devices. Even if it is possible to implement secure TRNG in user logic [19], we decided that this represents a serious risk when high security level certification is required. For that reason, random numbers can be generated in three places on the platform: inside the user logic of the FPGA, in the secure microprocessor, or in the smart card of the board.
- The secure microprocessor we selected provides many security features such as internal and external sensor management. A battery powered key memory is automatically erased when an attack on the clock, the package of the chip, the temperature, or the power supply is detected. The package integrity sensor is very important since FPGA vendors do not provide such packages. Thanks to this feature it is possible to produce a low-cost hardware security module without surface enclosure, where the heart of the security level (the secure microprocessor) is physically protected by its package.
- The cost of such a microprocessor is very low compared to the FPGA. PCB complexity increases with this BGA device, but this cost should be compared with the non-negligible security add on.

Based on this choice, the secure microprocessor will be the heart of the system since it generates cryptographic keys and stores the master key.

FPGA Architecture The goal is not to develop a fixed architecture but a flexible one that allows the design to be scaled according to application requirements. For instance, an application that requires many RSA operations in parallel will have several embedded RSA hardware accelerators in order to achieve higher throughput. The configuration of the platform is currently performed off line, before physical synthesis of the FPGA architecture. Ultimately the goal is to provide run-time reconfiguration according to application requirements, by using dynamic reconfiguration. This goal was not addressed in this study but will the subject of future works in order to provide such functionality while keeping the same level of security. Since cryptographic operational keys are stored in the secure microprocessor of the board, the communication between the FPGA and this component must be secured. This constraint leads to the architecture described in Fig. 6.23.

Figure 6.23 shows the architecture we developed. The "core Wrapper" square refers to a slot where the platform user can instantiate any IP cores compatible with the open source wishbone bus [33], these may be cryptographic IP cores or even application specific IPs. To make the platform usable from the outside world, it is equipped with PCI Express connectivity.

A direct connection exists from the secure processor and key memories. These memories can only be written from the microprocessor, so with this FPGA architecture, the key cannot travel to the host. An AES decryption engine is placed between

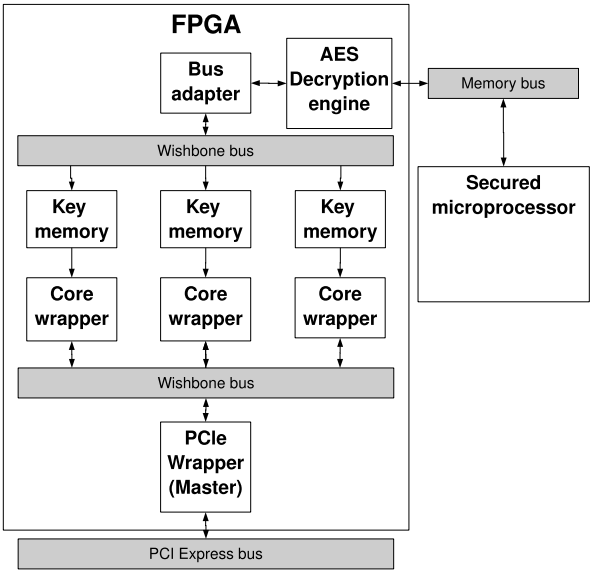


Fig. 6.23 General target architecture of the reconfigurable cryptographic processor

the external processor and the key memory to prevent board level attacks (for instance using bus probing).

High Speed Communication Interfaces To be as flexible as possible according to Virtex5 functionalities, we decided to connect the FPGA to a PCI Express bus with eight lines with a maximum throughput of 16 giga bits per second on each of the two ways (reception and transmission). We also connected the FPGA to two RJ-45 interfaces that can support giga bit Ethernet, we deliberately chose two interfaces to allow red/black architecture [41] that are commonly used for governmental or military applications.

Trusted Area A secure microprocessor allows tamper respondent mechanisms to fulfill FIPS140-2 level 3 requirements; however Virtex5 FPGA cannot comply with this standard since its package does not respond to tampering and so physical invasive attacks are feasible. To allow our secure platform to reach level 3, we designed the board's PCB to facilitate the use of a tamper resistant enclosure such as [23]. This was achieved simply by reducing the area to be protected to the minimum in order to decrease cost of enclosure. The secure area boundary is shown in Fig. 6.24, it includes:

- The FPGA where user keys are used;
- The secure microprocessor where master key is stored and user keys are generated;
- The FPGA configuration flash memory, since we have shown that bitstream security is not totally guaranteed by FPGA vendors (bitstream replay, bitstream integrity);

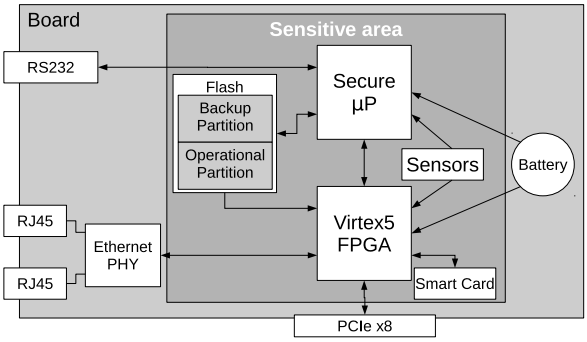


Fig. 6.24 General scheme of the NETHEOS platform

- The core voltage regulator of the FPGA, this increases the robustness of the platform regarding power analysis because an attacker can only measure current filtered by the regulator;
- A smart card used to diversify true random sources (from the smart card, the secure processor and the FPGA);
- And finally the sensors used to monitor enclosure integrity, allowing the secure microprocessor to erase the master key.

Security Scalability To achieve security level scalability of the board, two types of attacks have to be distinguished:

- Logical attacks that must be countered by implementing a logically secured hardware and software design.
- Physical attacks that can be prevented by implementing cryptographic algorithms with side channel and fault injection countermeasures, and also by adding a secure surface enclosure.

Tamper respondent enclosure is widely used for hardware security modules (HSM) that erase cryptographic keys when an attack is detected, thus ensuring protection against invasive attacks that are quasi impossible to counter only with logical mechanisms. With such protection, this platform should achieve FIPS 140-2 certification at level 3 where key destruction is required to counter attacks. Secure surface enclosure is an industrial process that consists in enclosing a security system inside a tamper respondent envelope. It is an active seal that generates an alarm when the system is tampered with; this alarm can be used by the system to erase sensitive data. These surface enclosures thus have to be powered to enable the system (which requires energy) to erase the keys. That is why most HSMs include a battery, which can be located inside or outside the secure enclosure, since if the attacker simply removes the battery, the keys will be erased as they are stored in a volatile memory. Development of a secure enclosure is a hard task, particularly if a high level of security is required. The classical approach is to enclose the system in a mesh of conductive wires. This mesh is connected to sensors that detect variations in wire conductivity and trigger alarms. Therefore, if attackers try to remove the protection, they are very likely to cut a wire. However an attacker with a high degree of

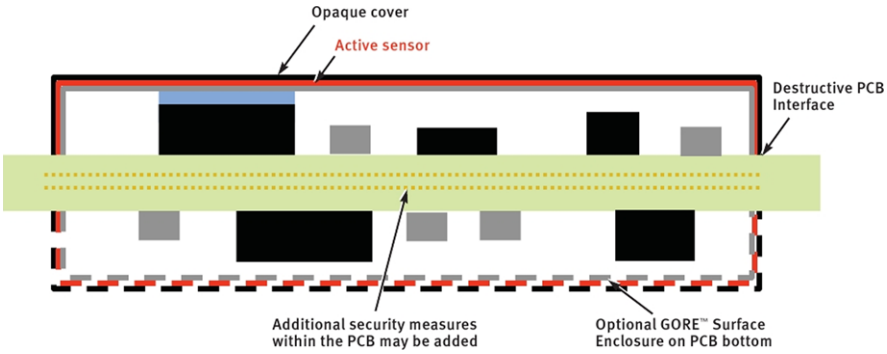


Fig. 6.25 Secure PCB surface enclosure designed by Gore

competency and sophisticated equipment may localize the mesh by using X-ray and succeed in passing between the wires without cutting them. A company called Gore proposes surface enclosures that have already been tested and validated, hence reducing the cost for a company that wants to create a physically protected device. The conductive mesh used by Gore is non-metallic to avoid X-ray attacks. The principle of the architecture of a Gore secure surface enclosure is shown in Fig. 6.25 [23]. This figure shows the PCB of the system enclosed by the Gore active sensor and an opaque cover. The cover is connected to the PCB by a destructive interface that triggers an alarm when tampered with.

The Gore tamper respondent sensor (in gray in the figure) can enclose the whole system or a part of the PCB. In the latter case, contacts between the enclosure and PCB require particular attention since the assailant may try to attack this weak part. That is why additional security mechanisms can be added inside the PCB itself. An opaque cover, generally made of metal, is added to protect the system from the environment. The board developed during the present study enables the use of such tamper respondent enclosures. It includes a battery and circuitry to allow key erasure. Moreover, since data remanence of sensitive information is obviously a concern for the final device, a mechanism that connects power to ground is also included.

Secure Key Management The secure microprocessor stores the value of a master key in its secure battery powered key memory, and operational keys are encrypted using this master value. Since the FPGA is the device that performs high-speed cryptography using dedicated hardwired engines, the secure microprocessor must decrypt operational key and send the plaintext value to the FPGA. The communication channel between FPGA and microprocessor can be secured using encryption according to security objectives of the application.

Secure Backup/Restore Operational key backup and restoration operations are performed using a safepad and a smart card. The safepad ensures there is a trusted path to enter a PIN code (without passing through the host computer), the smart card ensures the physical security of the backup.

Communication Between FPGA and Secure Processor The secure microprocessor is the device that generates and securely stores user objects. The FPGA device is used to accelerate cryptographic computations, consequently it has to use these cryptographic keys in plaintext form. However unencrypted communications between the FPGA and the processor are not acceptable when local attacks are possible, an attacker can spy on the communication bus to retrieve cryptographic keys. Even with a tamper respondent enclosure, it is preferable to encrypt the communication channel between the two chips so that all security does not rely on the enclosure. In order to communicate safely, the two chips have to share a symmetric key; this cannot be the master key because that key must never leave the secure microprocessor, otherwise the confidentiality of this key is harder to ensure. Thus a mechanism of key sharing must be implemented. At least two methods can be used to reach this goal:

- During the bitstream and processor code generation process the system designer hard codes a symmetric key in the FPGA logic and in the flash memory of the processor. There is no need for a key sharing protocol. However this method has certain drawbacks, first, the system designer knows the symmetric key value and the customer does not necessarily trust the SD. And, in order to simplify key management, bitstream and processor code generation process, the SD should provide the same symmetric key to all his designs, therefore breaking one product enables all the products to be broken. Finally the shared key remains the same throughout the life of the product, whereas side channel attacks such as DPA can lead to key discovery; however, if the key is changed regularly, power analysis attacks become more difficult.
- The system designer hard codes a private key pair in the processor code and another key pair in the FPGA bitstream, both signed by a trusted authority to avoid man in the middle attacks. These key pairs will be used to regularly exchange symmetric keys to avoid differential side channel attacks (DPA, DEMA). However this method has the same drawbacks as the previous one, and the cost of implementation is higher since asymmetric and symmetric algorithms have to be used.

6.5.3 Performance Results

In order to measure real performance and to show the advantage of having many IPs that can operate in parallel, an openSSL engine [34] was developed. The term engine here refers to a library that is dynamically linked to openSSL and allows this application to perform cryptographic operations using hardware devices. The engine developed is quite simple, it is based on the GMP (GNU Multi Precision) engine, i.e. the software implementation of the cryptographic algorithm that uses GMP, whereas openSSL uses a library called *Big Numbers*. This option was preferred since the GMP engine code is quite simple to modify. The code that performs the modular exponent in software has been replaced by a call to the RSA IP

Key size	RCP (-multi 4)	AMD Sempron 3000+ (-multi 1)	Intel Core2 CPU 2.13 GHz (-multi 2)
512	1282 sign/s	1579 sign/s	3334 sign/s
1024	421 sign/s	360 sign/s	731 sign/s
2048	85 sign/s	66 sign/s	134 sign/s
4096	13 sign/s	10 sign/s	22 sign/s

Fig. 6.26 Practical results of openSSL engine with CRT using six RSA IP cores

cores connected to the platform. GMP uses by default the Chinese remainder theorem (CRT), which allows the RSA exponent to be split into two smaller exponents, thereby dramatically reducing modular exponent computation time. For instance a 1024-bit modular exponent is reduced to two 512-bit modular exponents. This engine measures performance and checks the results of RSA computation using a simple call to the command line:

```
openssl speed -multi X -engine gmp rsa
```

The *multi* argument of the command line allows X RSA computation to be run in parallel, for instance a call with *-multi 2* gives better results when the test runs on a dual core processor. For the platform, the best results are achieved when the multiple parallel operation number is equal to the number of RSA IP cores instantiated. For instance, results of a test on a platform that includes six RSA IP cores are given in Table 6.26.

The results of openSSL engines were compared to two different types of general purpose CPUs. A single core processor: AMD Sempron, whose best performance is obviously achieved using the argument *-multi 1* while the Intel Core 2 processor allows parallel RSA computation giving better results with *-multi 2*. The best results were achieved with the platform using *-multi 4* because the CRT algorithm is used, and consequently two RSA engines are needed to perform one complete exponent. In terms of efficiency, a good way to compare is to sum the costs of the solutions. An Intel Core 2 processor running at 2 GHz cost about US$ 70 at the time of writing, while a Virtex5 LX30T cost about US$ 350, i.e. a ratio of five to one, while performance was twice lower on our implementation than on Virtex5 LX30T. So with our RSA engine, FPGAs are not competitive in terms of the cost: performance ratio using a general purpose CPU. However when the platform is used for massive RSA computation, the host CPU usage remains low. The most important argument for our FPGA-based implementation, is that by using the platform, it is possible to enhance the physical security of RSA private keys since they can be generated on board, used inside the platform, and never travel through host memory.

6.5.4 Conclusions

According to the results of the present study, there is no perfect FPGA device currently available for security-sensitive or cryptographic applications. Even the choice

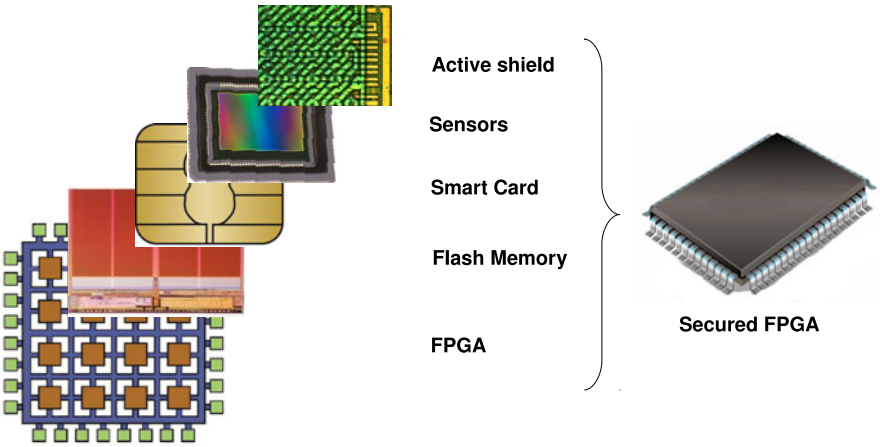


Fig. 6.27 An imaginary highly secured FPGA

between non-volatile and volatile FPGAs is not trivial and also often depends on the application. For instance, we were forced to use an external processor, which complicates the PCB design, increases vulnerabilities of the communication with the FPGA and represents an additional cost. Therefore, in this conclusion we 'imagine' a perfect FPGA device for such applications (see also Fig. 6.27):

- First it must have some embedded secure battery powered memories for highly sensitive data that can be very rapidly erased from user logic or by an external alarm.
- It must also include non-volatile memories that are useful in security, for instance to implement PIN code mechanisms or to store bitstream version TAGs to prevent bitstream replay attacks.
- FPGA vendors must provide strong mechanisms that ensure bitstream confidentiality and integrity, and that prevent bitstream replay attacks.
- The device must have an embedded mechanism allowing the system designer to force the FPGA device to accept only encrypted and authenticated bitstream, like the Actel Flash Lock mechanism or the Xilinx e-Fuse.
- Ideally it should have an embedded certified random number generator, such as a smart card. An even better solution would be to embed a smart card in the same package as the FPGA matrix.
- The FPGA device should be embedded inside a tamper proof or even better a tamper respondent package that triggers alarms when attacks are detected. For instance by using a metal mesh able to detect package opening and to trigger built in mechanisms to erase a battery powered memory. This package should also provide basic protection against electromagnetic analyses such as a Faraday cage.

Future works will focus on improving the platform by adding run time reconfiguration based on application requirements and by adding new cryptographic IP cores or by improving the performance of existing ones as well as improving communication architecture.

References

1. The Embedded Microprocessor Benchmark Consortium (EEMBC)
2. Fips Pub 197: "Advanced Encryption Standard (AES)" (2001)
3. In-System Programming (ISP) of Actel low-power flash devices using Flashpro3. Microsemi (ex Actel) corporation. Version 1.5 (August 2009)
4. Actel: Implementation of security in Actel's ProASIC and ProASICPLUS flash-based FPGAs. URL. http://www.actel.com/documents/Flash_Security_AN.pdf (2003)
5. Actel: Fusion FPGA Handbook. URL. http://www.actel.com/documents/Fusion_HB.pdf (2008)
6. Actel: ProASIC3 Handbook. URL. http://www.actel.com/documents/PA3_HB.pdf (2008)
7. Alderighi, M., D'Angelo, S., Mancini, M., Sechi, G.R.: A fault injection tool for SRAM-based FPGAs. In: IEEE International On-Line Testing Symposium, p. 129 (2003). doi:10.1109/OLT.2003.1214379
8. Altera: Design Security in Stratix III Devices. URL. http://www.altera.com/literature/wp/wp-01010.pdf (2006)
9. Badrignans, B., Champagne, D., Elbaz, R., Gebotys, C.H., Torres, L.: Sarfum: security architecture for remote FPGA update and monitoring. ACM Trans. Reconfigurable Technol. Syst. **3**(2), 8 (2010)
10. Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: Annual IEEE Symposium on Foundations of Computer Science pp. 90–99 (1991). doi:10.1109/SFCS.1991.185352
11. Daemen, J., Rijmen, V.: AES proposal: Rijndael (1998)
12. Drimer, S.: Authentication of FPGA bitstreams: why and how. In: Applied Reconfigurable Computing. Lecture Notes in Computer Science, vol. 4419, pp. 73–84 (2007)
13. Drimer, S.: Volatile FPGA design security—a survey (v0.96) (April 2008)
14. Eisenbarth, T., Güneysu, T., Paar, C., Sadeghi, A.-R., Schellekens, D., Wolf, M.: Reconfigurable trusted computing in hardware. In: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing, STC '07, pp. 15–20. ACM, New York (2007). doi:10.1145/1314354.1314360
15. Elbaz, R.: Hardware mechanisms for secured processor memory transactions in embedded systems. PhD thesis, University of Montpellier (December 2006)
16. Elbaz, R., Torres, L., Sassatelli, G., Guillemin, P., Bardouillet, M., Martinez, A.: A comparison of two approaches providing data encryption and authentication on a processor memory bus. In: Vounckx, J., Azemard, N., Maurine, P. (eds.) Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation. Lecture Notes in Computer Science, vol. 4148, pp. 267–279. Springer, Berlin (2006). doi:10.1007/11847083_26
17. Elbaz, R., Torres, L., Sassatelli, G., Guillemin, P., Bardouillet, M., Martinez, A.: A parallelized way to provide data encryption and integrity checking on a processor-memory bus. In: Proceedings of the 43rd Annual Design Automation Conference, DAC '06, pp. 506–509. ACM, New York (2006). doi:10.1145/1146909.1147042
18. Elbaz, R., Champagne, D., Lee, R., Torres, L., Sassatelli, G., Guillemin, P.: TEC-tree: a low-cost, parallelizable tree for efficient defense against memory replay attacks. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems—CHES 2007. Lecture Notes in Computer Science, vol. 4727, pp. 289–302. Springer, Berlin (2007). doi:10.1007/978-3-540-74735-2_20

19. Fischer, V., Bernard, F., Bochard, N., Varchola, M.: Enhancing security of ring oscillator-based RNG implemented in FPGA. In: Field-Programable Logic and Applications (FPL), September, pp. 245–250 (2008)
20. Fruhwirth, C.: New methods in hard disk encryption. Technical report, Institute for Computer Languages, Theory and Logic Group, Vienna University of Technology (2005)
21. Gaj, K., Chodowiec, P.: Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays (2001)
22. Gassend, B., Suh, G.E., Clarke, D., van Dijk, M., Devadas, S.: Caches and Merkle trees for efficient memory integrity verification. In: Proceedings of Ninth International Symposium on High Performance Computer Architecture, February (2003)
23. GORE: GORE tamper respondent surface enclosure. Commercial Brochure (2007)
24. Hall, W.E., Jutla, C.S.: Parallelizable authentication trees. In: Selected Areas in Cryptography, pp. 95–109 (2005)
25. Hendry, M.: Multi-application Smart Cards: Technology and Applications. Cambridge University Press, Cambridge (2007).
26. Hori, Y., Satoh, A., Sakane, H., Toda, K.: Bitstream encryption and authentication using AES-GCM in dynamically reconfigurable systems. In: Matsuura, K., Fujisaki, E. (eds.) Advances in Information and Computer Security. Lecture Notes in Computer Science, vol. 5312, pp. 261–278. Springer, Berlin (2008). doi:10.1007/978-3-540-89598-5_18
27. Lattice: LatticeXP2 Family Handbook. URL. http://www.latticesemi.com/dynamic/view_document.cfm?document_id=24315 (2008)
28. Lie, D., Thekkath, C.A., Horowitz, M.: Implementing an untrusted operating system on trusted hardware. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03, pp. 178–192. ACM, New York (2003). doi:10.1145/945445.945463
29. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography, 1st edn. CRC Press, Boca Raton (1996). ISBN 0849385237
30. Merkle, R.C.: Protocols for public key cryptography. In: Proceedings of IEEE Symp. on Security and Privacy, pp. 122–134 (1980)
31. Netheos: Official Netheos Website. URL. http://www.netheos.net (2010)
32. Note, J.-B., Rannaud, E.: From the bitstream to the netlist. In: Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays, FPGA '08, pp. 264–264. ACM, New York (2008). doi:10.1145/1344671.1344729
33. OpenCores.org: WISHBONE system-on-chip interconnection architecture for portable IP cores specification revision B.3. URL. www.opencores.org/downloads/wbspec_b3.pdf (2002)
34. OpenSSL.org: OpenSSL cryptography and SSL/TLS toolkit, engine documentation. URL. http://www.openssl.org/docs/crypto/engine.html
35. Parelkar, M.M., Gaj, K.: Implementation of EAX mode of operation for FPGA bitstream encryption and authentication. In: Brebner, G.J., Chakraborty, S., Wong, W.-F. (eds.) FPT, pp. 335–336. IEEE Press, Singapore (2005)
36. Schellekens, D., Tuyls, P., Preneel, B.: Embedded trusted computing with authenticated non-volatile memory. In: Proceedings of the 1st International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing—Challenges and Applications, Trust '08, pp. 60–74. Springer, Berlin (2008). doi:10.1007/978-3-540-68979-9_5
37. Suh, G.E., O'Donnell, C.W., Devadas, S.: AEGIS: a single-chip secure processor. IEEE Des. Test **24**, 570–580 (2007). doi:10.1109/MDT.2007.179
38. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th Annual International Conference on Supercomputing, ICS '03, pp. 160–171. ACM, New York (2003). doi:10.1145/782814.782838
39. Suh, G.E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: Efficient memory integrity verification and encryption for secure processors. In: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 36, p. 339. IEEE Comput. Soc., Washington (2003)

40. Suh, G.E., O'Donnell, C.W., Sachdev, I., Devadas, S.: Design and implementation of the AEGIS single-chip secure processor using physical random functions. In: Proceedings of the 32nd Annual International Symposium on Computer Architecture, ISCA '05, pp. 25–36. IEEE Comput. Soc., Washington (2005). doi:10.1109/ISCA.2005.22
41. Thales: Security architecture reinforced in two or three physically separated sections. Commercial Brochure (2007)
42. Unknown: Open freebox project website. URL. http://www.f-x.fr/ (2006)
43. Xilinx: Virtex-5 FPGA configuration user guide. URL. http://www.xilinx.com/support/documentation/user_guides/ug191.pdf (2008)
44. Xilinx: Xilinx Ug360 Virtex-6 FPGA configuration user guide. URL. www.xilinx.com/support/documentation/user_guides/ug360.pdf (2010)
45. Yang, J., Zhang, Y., Gao, L.: Fast secure processor for inhibiting software piracy and tampering. In: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 36, p. 351. IEEE Comput. Soc., Washington (2003)

Chapter 7
Conclusions

Benoit Badrignans, Jean-Luc Danger, Guy Gogniat, Viktor Fischer, and Lionel Torres

This book addresses a wide spectrum of techniques and solutions for reconfigurable platforms. Most of the techniques can be also used or adapted for other domains (for instance System on Chip) but here we chose to discuss FPGAs in detail, as they are increasingly used for many embedded systems applications. The security analysis described at the beginning of the book identified links between technology, architecture and systems. We showed that at the technology level, it is possible to offer new logic styles to prevent side channel attacks, and, at the same time, dedicated FPGA designs that respond to these attacks. Another key point is undoubtedly how to generate encryption keys on an FPGA. This is crucial for the robustness of a complete system that can support the generation and management of protected sessions. We also showed that it is possible to compare these techniques and we assessed their relative effectiveness. Finally, we discussed how to manage the confidentiality and integrity of data and code of a reconfigurable platform, including providing hardware protection mechanisms. These safeguards also protect the bitstream configuration of an FPGA. Research in this domain is not yet advanced, despite the fact it is indispensable for the security of re-configurable platforms. For the future, modern technology offers users easy access to many widely used global networks (WiFi, 3G data networks, LAN, etc.) and hassle free access to private services, data management and storage. For instance, Google services offer remote storage with gigabytes of storage, enabling users to access their private data (documents, agenda, contacts, etc.) from any location using any connected device (laptop, cell phone, PDA). This may not be completely true for everyone, but is undoubtedly representative of the short term future. In the first half of the 20th century, wired home phones were rare, whereas today almost everyone owns at least one cell phone. This trend, known as Pervasive or Ubiquitous computing, belongs to the post desktop era

L. Torres (✉)
LIRMM—UMR CNRS 5506, University of Montpellier 2, Montpellier, France
e-mail: lionel.torres@lirmm.fr

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3_7, © Springer Science+Business Media B.V. 2011

of human-machine interaction in which information processing has penetrated everyday objects and activities to such an extent that many ordinary activities openly incorporate computational devices of different types. For example, a home ubiquitous computing environment might interconnect lighting and environmental controls with personal biometric monitors woven into clothing so that illumination and heating conditions are remotely continuously and imperceptibly adjusted. Pervasive computing generally encourages the massive digitization of personal information and services and draws attention to the security issues involved in protecting privacy and intellectual property (IP). The wide spectrum of attacks and consequences described in such scenarios not only concerns a well identified set of information and services, but may also compromise other aspects ranging from environmental control to home security. Flexibility will be the key for all new paradigm computing approaches in the coming years. The use of reconfigurable platforms is already underway, and security concerns are pointed to as one of the main bottlenecks that need to be addressed. The aim of this book was to give readers and security engineers an overview of a wide spectrum of current solutions. Throughout the book, our objective was to present some hardware security schemes that match the requirements of embedded systems in order to protect the data and the code contained in reconfigurable platforms.

Glossary

ADC Analog Digital Converter.
AE Authenticated Encryption.
AES Advanced Encryption Standard.
ALU Arithmetic Logic Unit.
ANR Agence Nationale de la Recherche, French National Research Agency.
AREA Added Redundancy Explicit Authentication.
ANSI American National Standards Institute.
ASIC Application Specific Integrated Circuit.
BCDL Balanced Cell-based Differential Logic.
BGA Ball Grid Array.
BPA Branch Prediction Analysis.
BRAM Block Random Access Memory.
CASR Cellular Automata Shift Register.
CBC Cipher Block Chaining.
CMOS Complementary Metal Oxide Semiconductor.
CPA Correlation Power Analysis.
CPU Central Processing Unit.
CRC Cyclic Redundancy Code.
CRT Chinese Remainder Theorem.
CSP Critical Security Parameter.
DEMA Differential ElectroMagnetic Analysis.
DES Data Encryption Standard.
DFA Differential Fault Analysis.
DFF D Flip-Flop.
DLL Digital Locked Loop.
DoM Difference of Mean.
DoS Denial of Service.
DPA Differential Power Analysis.
DPL Dual rail Precharge Logic.
DRSL Dual-rail Random Switching Logic.
DWDDL Double WDDL.

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3, © Springer Science+Business Media B.V. 2011

ECB Electronic Code Book.
ECC Elliptic Curve Cryptographic.
EE Early Evaluation.
EEPROM Electrically Erasable Programmable Read-Only Memory.
EM ElectroMagnetic.
EMA ElectroMagnetic Analysis.
EMC ElectroMagnetic Compatibility.
EMI ElectroMagnetic Interference.
EPA Entropy Power Analysis.
FA Fault Attack.
FIB Focused Ion Beam.
FIFO First In First Out.
FIPS Federal Information Processing Standard.
FPGA Field Programmable Gate Array.
GCM Galois/Counter Mode.
GF Galois Field.
HDL Hardware Description Language.
HMAC Hash-based Message Authentication Code.
HO-DPA High Order Differential Power Analysis.
HRNG Hybrid Random Number Generator.
HSM Hardware Security Modules.
IC Integrated Circuit.
ICAP Internal Configuration Access Port.
ILP Instruction Level Parallelism.
IP Intellectual Property.
ISP Internet Service Protocol.
IPC Instruction Per Cycle.
IWDDL Isolated WDDL.
JTAG Joint Test Action Group.
LAN Local Area Network.
LFSR Linear Feedback Shift Register.
LRA Leak Resistant Arithmetic.
LSB Less Significant Bit.
LUT Look-Up Table.
MAC Message Authentication Code.
MAMU MAsked Memory Unit.
MDPL Masked Dual rail Precharge Logic.
MIA Mutual Information Analysis.
MMU Management Memory Unit.
MSB Most Significant Bit.
NIST National Institute of Standards and Technology.
NVM Non Volatile Memory.
OS Operating System.
OTP One Time Pad.
PCB Printed Circuit Board.

PCI Peripheral Component Interconnect.
PDA Personal Digital Assistant.
PDF Probability Density Function.
PE-ICE Parallelized Encryption and Integrity Checking Engine.
PLL Phase Locked Loop.
PPA Partitioning Power Analysis.
PRNG Pseudo Random Number Generator.
PTI PlainText Input.
RAM Random Access Memory.
RISC Reduced Instruction Set Computer.
RNG Random Number Generator.
RNS Residue Number System.
ROM Read Only Memory.
RSA Rivest, Shamir and Adleman.
RTL Register Transfer Level.
SAFES Security Architecture for Embedded Systems.
SCA Side Channel Attack.
SD System Designer.
SEMA Simple ElectroMagnetic Analysis.
SEP Security Executive Processor.
SHA Secure Hash Algorithm.
SMM Security Memory Map.
SNR Signal Noise Ratio.
SoC System on Chip.
SPA Simple Power Analysis.
SRAM Static Random Access Memory.
SSL Secure Sockets Layer.
STTL Secure Triple Track Logic.
TA Template Attack.
TASC Time Address Segment Cipher.
TB Technological Bias.
TEC-Tree Tamper Evident Counter Tree.
TERO Transition Effect Ring Oscillator.
TPM Trusted Platform Module.
TRNG True Random Number Generator.
TS Trusted Party or Time Stamp.
USB Universal Serial Bus.
USM Universal Sbox Masking.
VPA Variance Power Analysis.
VPN Virtual Private Network.
WDDL Wave Dynamic Differential Logic.
XOM eXecute Only Memory.

Index

A
AES, 7, 8, 25, 26, 32–36, 39, 41, 42, 51, 52, 60, 61, 77, 79, 92, 93, 96, 146, 148–150, 152, 154, 155, 159, 160, 165, 167, 174, 176–178, 191
ASIC, 2, 8, 13, 22–26, 30–32, 43, 73, 74, 89, 93–96, 105, 109, 127, 129, 137, 140, 157, 191

B
BCDL, 32, 74, 87, 89, 91–96, 191
Bitstream, 3, 10, 14–16, 24, 25, 29, 30, 38, 48, 103, 113, 137, 138, 140, 141, 154–174, 176–179, 182, 184, 189

C
Confidentiality, 2, 3, 5, 9, 13, 14, 22, 39, 43, 141, 147, 148, 150, 151, 153–155, 158, 160, 165, 168, 170, 171, 173, 174, 182, 184, 189
Countermeasure, 2, 5, 9, 12, 15, 22, 26, 30–32, 34, 52, 60, 61, 64, 68, 73–77, 82, 85, 86, 94–96, 101, 141, 175, 176, 180
CPA, 26, 65, 67, 191
CSP, 17–21, 191

D
DEMA, 27, 28, 38, 55, 59, 60, 63, 68, 85, 86, 182, 191
DES, 26–28, 52, 61–65, 68, 78–83, 85, 86, 170, 171, 173, 191
DLL, 109, 191
DPA, 4, 10, 16, 25, 32, 33, 38, 59–65, 67, 68, 77, 182, 191
DPL, 31, 32, 87, 92–95, 191
DRSL, 94, 95, 191

E
EM, 56, 58–63, 68, 192
EMC, 19, 21, 192
EMI, 19, 21, 117, 192
EPA, 68, 78, 83, 192

F
FIPS, 2, 9, 17, 18, 20, 21, 24, 38, 43, 111, 175, 180, 192

H
Hamming, 58, 59, 62–64, 76, 77, 81
Hardware attack, 10, 11
Hiding, 31, 74, 75, 85, 86, 95, 96
HO-DPA, 68, 77, 78, 192
HRNG, 102, 192

I
Integrity, 2, 3, 5, 9, 14, 21, 22, 30, 39, 40, 43, 141, 143–151, 153–156, 159–163, 165, 168, 170, 171, 173, 174, 176, 178–180, 184, 189, 193
Invasive, 4, 5, 23, 24, 38, 39, 48, 50, 51, 157, 175, 179, 180

J
Jitter, 75, 104, 114–124, 127, 128, 130

K
Key, 2, 4, 5, 7–14, 16, 17, 19–23, 25–32, 34–36, 38–40, 47, 48, 51, 52, 56, 59–68, 74, 75, 78, 80, 82, 86, 91, 101, 102, 106, 107, 111, 113, 132, 137–139, 143, 146, 148, 149, 153, 155, 157–161, 163–165, 167–171, 174–183, 189, 190

B. Badrignans et al. (eds.), *Security Trends for FPGAS*,
DOI 10.1007/978-94-007-1338-3, © Springer Science+Business Media B.V. 2011

L
LRA, 33, 34, 192

M
MAC, 143–145, 149, 159, 167, 170, 192
Masked, 32, 68, 75–82, 84, 85, 93, 103, 192
MDPL, 32, 93–95, 192
MIA, 65, 67, 68, 78, 192

N
NIST, 8, 111, 126, 149, 192

O
Oscillator, 104, 109, 114, 115, 118–122, 124–127, 129, 130, 193
OTP, 41, 42, 146–148, 192

P
PE-ICE, 39–41, 148–153, 193
Physical attack, 16, 31, 138, 141, 161, 166, 175, 180
PLL, 104, 109, 122–124, 193
Probe, 23, 24, 29, 54, 55, 57, 157
Processor, 8, 9, 11, 13, 14, 16, 19, 27, 36, 38–42, 84, 85, 95, 137, 139–145, 148, 150, 154, 164, 178–180, 182–184, 193
PTI, 56, 60–63, 65, 193

R
Random number generators, 101
  HRNG, 102
  PRNG, 102
  TRNG, 102
Remote configuration, 13, 30, 36, 163, 167, 172
Replay, 3, 40, 142–146, 149, 152–155, 161–168, 170, 171, 173, 174, 179, 184
RNG, 20, 95, 101, 102, 159, 193
RNS, 33, 34, 193
RSA, 7, 8, 26, 34, 52, 59, 60, 138, 174, 176–178, 182, 183, 193

S
SAFES, 36, 37, 193
SCA, 5, 27, 51, 54, 58, 64–66, 68, 74, 75, 84, 94, 95, 193
SEMA, 59, 60, 68, 193
Sensor, 23, 31, 39, 52, 55, 56, 58, 176, 178, 180, 181
SEP, 38, 193
Side channel attacks, 5, 16, 20, 25, 34, 51, 52, 56, 68, 73, 74, 82, 101, 107, 141, 159, 182, 189
Software attacks, 11, 12
SPA, 4, 10, 16, 25, 59, 60, 68, 193
Splicing, 40, 142–145, 149, 154
Spoofing, 40, 41, 142, 143, 149, 154, 155, 162
STTL, 32, 74, 87, 89, 90, 95, 193
System designer, 4, 5, 140, 141, 158–161, 163–166, 168, 174, 182, 184, 193
System owner, 4, 5, 140

T
TASC, 148, 154, 155, 193
Template attack, 59, 66–68, 193
Threat, 1–5, 11, 12, 15, 22, 29, 30, 36, 38, 43, 80, 85, 137–139, 141, 143, 145, 157, 159, 162, 166, 167, 173, 176
Tree, 143, 145–148, 152–154, 193
TRNG, 102–105, 107–114, 119–121, 123–126, 128–132, 178, 193
Trusted platform, 162, 193

U
USM, 80–83, 193

V
VPA, 68, 78, 83, 193

W
WDDL, 32, 74, 87–89, 92–95, 191–193