Remote Hardware Attacks on Connected Devices

Joseph Gravellier

N° d'ordre NNT: *2021LYSEM034*

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein du
Laboratoire Systèmes et Architectures Sécurisés et Thales

Ecole Doctorale N°488
Science, Ingéniérie, Santé

Microélectronique / Systèmes Embarqués Sécurisés

Soutenue publiquement le 03/12/2021 par:

Joseph Gravellier

Remote Hardware Attacks on Connected Devices

Devant le jury composé de:

Edith KUSSENER, Professeur, *ISEN*	Présidente / Rapportrice
Régis LEVEUGLE, Professeur, *Grenoble INP, Université Grenoble Alpes*	Rapporteur
Jacques FOURNIER, Chef du laboratoire de sécurité, *CEA-Leti*	Examinateur
Emmanuel PROUFF, Chef adjoint de la division Produits et Services Sécurisés, *ANSSI*	Examinateur
Lilian BOSSUET, Professeur, *LaHC/Université Jean Monnet*	Examinateur
Jean-Max DUTERTRE, Professeur, *Mines St Etienne*	Directeur de thèse
Yannick TEGLIA, Ingénieur expert sécurité matérielle, *Thales*	Encadrant
Philippe LOUBET MOUNDI, Manager expert sécurité matérielle, *Thales*	Encadrant

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	M. Batton-Hubert
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	N. Absi, Maitre de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	V.Laforest, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

ABSI	Nabil	MR	Génie industriel	CMP
AUGUSTO	Vincent	MR	Génie industriel	CIS
AVRIL	Stéphane	PR	Mécanique et ingénierie	CIS
BADEL	Pierre	PR	Mécanique et ingénierie	CIS
BALBO	Flavien	PR	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR	Mathématiques appliquées	FAYOL
BEIGBEDER	Michel	MA	Informatique	FAYOL
BILAL	Blayac	DR	Sciences et génie de l'environnement	SPIN
BLAYAC	Sylvain	PR	Microélectronique	CMP
BOISSIER	Olivier	PR	Informatique	FAYOL
BONNEFOY	Olivier	PR	Génie des Procédés	SPIN
BORBELY	Andras	DR	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR	Génie Industriel	FAYOL
BRUCHON	Julien	PR	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	PR	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR	Génie Industriel	CMP
DEBAYLE	Johan	MR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA	Génie industriel	Fayol
DELAFOSSE	David	PR	Sciences et génie des matériaux	SMS
DELORME	Xavier	PR	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
BERGER-DOUCE	Sandrine	PR	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR	Mécanique et ingénierie	SMS
DUTERTRE	Jean-Max	PR	Microélectronique	CMP
EL MRABET	Nadia	MA	Microélectronique	CMP
FAUCHEU	Jenny	MA	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	MR	Génie des Procédés	SPIN
FEILLET	Dominique	PR	Génie Industriel	CMP
FOREST	Valérie	PR	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GAVET	Yann	MA	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA	Sciences et génie des matériaux	CIS
GONDRAN	Natacha	MA	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GRIMAUD	Frederic	EC	Génie mathématiques et industriel	FAYOL
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR	Génie des Procédés	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR	Génie des Procédés	SPIN
ISMAILOVA	Esma	MC	Microélectronique	CMP
KERMOUCHE	Guillaume	PR	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	DR	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	DR	Mécanique et ingénierie	FAYOL
LIOTIER	Pierre-Jacques	MA	Mécanique et ingénierie	SMS
MEDINI	Khaled	EC	Sciences et génie de l'environnement	FAYOL
MOLIMARD	Jérôme	PR	Mécanique et ingénierie	CIS
MOULIN	Nicolas	MA	Mécanique et ingénierie	SMS
MOUTTE	Jacques	MR	Génie des Procédés	SPIN
NAVARRO	Laurent	MR	Mécanique et ingénierie	CIS
NEUBERT	Gilles	PR	Génie industriel	FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
O CONNOR	Rodney Philip	PR	Microélectronique	CMP
PICARD	Gauthier	PR	Informatique	FAYOL
PINOLI	Jean Charles	PR	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	DR	Génie des Procédés	CIS
ROUSSY	Agnès	MA	Microélectronique	CMP
SANAUR	Sébastien	MA	Microélectronique	CMP
SERRIS	Eric	IRD	Génie des Procédés	FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
VALDIVIESO	François	PR	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR	Génie industriel	CIS
YUGMA	Gallian	MR	Génie industriel	CMP

Mise à jour : 04/02/2021

ABSTRACT

To meet the ever-growing need for performance in silicon devices, integrated-circuit providers have been increasingly relying on software-hardware cooperation. By controlling hardware resources such as power or clock management from the software, developers earn the possibility to build more flexible and power efficient applications. Despite the benefits, these hardware components are now exposed to software code and can potentially be misused as open-doors to new types of attacks.

In this thesis we aim at evaluating software-based hardware attacks, a novel attack family that targets connected devices such as IoT products, smartphones or cloud datacenters. These attacks take advantage of hardware resources that can be directly accessed from software in complex integrated-circuits and use them to conduct fault injection and side-channel analysis exploits. Because they are triggered by software code, they can be launched remotely and on multiple victim devices regardless of their physical location. Through the evaluation of software-based hardware attacks we aim at assessing the level of threat they pose to connected devices security and at providing countermeasure guidelines for building resistant systems. The challenge is considerable since any connected device is potentially endangered.

This manuscript gathers the research work conducted during this thesis to identify attack vectors in FPGAs and complex SoC systems. Through our experiments we discovered generic software-based hardware attack vectors widely implemented in recent SoCs that could enable remote hardware attacks. We conducted FPGA-to-FPGA and FPGA-to-CPU attacks and demonstrated that remote power side-channel analysis was feasible. These new attack paths represent a serious threat for FPGA-based systems especially when applied to multi-user cloud FPGAs and heterogeneous SoCs. Then, we went further by proving that delay-line components, widely implemented in high-end SoCs, could be used for conducting fault injection and side-channel attacks. We built various scenarios such as CPU-to-CPU and CPU-to-MCU attacks on Linux-based operating systems and demonstrated that software-based hardware attacks could successfully break the logical isolation between processes and lead to the extraction of sensitive information.

Throughout the chapters, we precisely describe the methodology adopted to build and conduct the attacks along with countermeasure proposals to mitigate their impact. Finally, according to our experimentations results and to other works published recently, it appears that any door left open for malwares to gain access to hardware resources could lead to full disclosure of a target's stored secrets. Therefore, there is an urgent need to build on the lessons learned from this thesis and employ countermeasures to effectively mitigate remote hardware attacks.

TABLE OF CONTENTS

Abstract iii

Table of Contents x

List of Figures xi

List of Tables xv

Glossary xvii

Avant Propos xxi

Résumé xxiii

1 Introduction 1
1.1 Thesis Context . . . 2
1.2 Thesis Objectives . . . 3
1.3 Roadmap: *From FPGA to CPU exploits* . . . 3
1.4 Contributions . . . 4
1.4.1 Software-based Power Analysis Attacks on FPGAs . . . 4
1.4.2 Software-based Power Analysis Attacks on Complex SoCs . . . 5
1.4.3 Software-based Fault Injection on SoC External Memory Transfers 5
1.5 Outline . . . 6

2 Background 7
2.1 Introduction to Hardware Attacks . . . 8
2.1.1 Origins . . . 8
2.1.2 Attack Classification . . . 8
2.1.3 Non-Invasive Attack Setup . . . 10
2.1.4 Non-Invasive Side-Channel Analysis Attacks . . . 11
2.1.4.1 Power Analysis Attacks . . . 11
2.1.4.2 Electromagnetic Analysis Attacks . . . 12
2.1.4.3 Other Side-Channels . . . 13
2.1.5 Non-Invasive Fault Injection Attacks . . . 14
2.1.5.1 Clock and Power Glitch Attacks . . . 14
2.1.5.2 Electromagnetic Glitch Attacks . . . 15
2.2 The Advent of Connected Devices . . . 16

2.2.1 Overview . . . 16
2.2.2 Applications and Threats . . . 16
2.2.3 Hardware Attacks: No Future? . . . 18
2.3 Remote Hardware Attacks . . . 18
2.3.1 The Origins of Remote Hardware Attacks . . . 18
2.3.1.1 The Increasing Complexity of Modern Devices . . . 19
2.3.1.2 The Adoption of Multi-User and Multi-Tasking Systems 20
2.3.1.3 The Emergence of Integrated Security . . . 21
2.3.2 Remote Hardware Attack Families . . . 24
2.3.2.1 TEMPEST Attacks . . . 24
2.3.2.2 Timing and Microarchitectural Attacks . . . 25
2.3.2.3 Software-based Hardware Attacks . . . 28
2.3.3 Software-based Hardware Attack Taxonomy . . . 29
2.3.4 Software-based Fault Injection Attacks . . . 30
2.3.4.1 Overview & Categorization . . . 30
2.3.4.2 Rowhammer-based Bit-flips Injection in DRAM Memories . . . 30
2.3.4.3 FPGA-based Power Glitch Injection . . . 32
2.3.4.4 DVFS-based Power/Clock Glitch Injection . . . 34
2.3.4.5 Delay-Line-based Glitch Injection on Memory Transfers 36
2.3.5 Software-based Side-Channel Analysis Attacks . . . 37
2.3.5.1 Overview & Categorization . . . 37
2.3.5.2 FPGAs-based Power Side-Channel Attacks . . . 37
2.3.5.3 ADC-based Power Side-Channel Attacks . . . 39
2.3.5.4 Delay-Line-based Power Side-Channel Attacks . . . 41
2.3.5.5 Intel RAPL-based Power Side-Channel Attacks . . . 41
2.3.6 Software-based Hardware Attack Privileges . . . 42
2.4 Conclusion . . . 45

3 Software-based Power Analysis Attacks on FPGAs 47
3.1 Chapter Introduction . . . 48
3.2 Technical Background . . . 50
3.2.1 FPGAs Voltage Fluctuations . . . 50
3.2.1.1 Power Supply Fluctuations . . . 50
3.2.1.2 Effect on Logic Propagation Delays . . . 51
3.2.2 Delay Sensors . . . 51
3.2.2.1 Time-to-Digital Converter-based Sensor . . . 52
3.2.2.2 Ring-Oscillator-based Sensor . . . 53
3.2.3 Threat Model: FPGA-based Power SCA . . . 53
3.2.4 Related Works: Existing Scenarios (2018) . . . 54
3.3 FPGA-to-FPGA - Designing High-Speed RO-based Sensors for FPGA-based SCA . . . 55
3.3.1 Introduction . . . 55

3.3.2 Motivation . . . 56
3.3.3 A Novel RO-based Sensor Design . . . 56
3.3.3.1 RO-based Sensors Downsides . . . 56
3.3.3.2 Designing a high frequency RO-based Sensor . . . 57
3.3.3.3 Number of RO-based sensors . . . 58
3.3.3.4 Place and Route Influence . . . 59
3.3.4 RO-sensor based Correlation Power analysis Attack . . . 59
3.3.4.1 Experimental Setup . . . 59
3.3.4.2 Correlation Power Analysis Model . . . 60
3.3.4.3 JRO-based Sensor CPA Results . . . 61
3.3.5 Further Results and Discussion . . . 62
3.3.5.1 TDC & Electromagnetic Experimental Setup . . . 62
3.3.5.2 Side-Channel Results . . . 62
3.3.5.3 Discussion . . . 63
3.3.6 Conclusion . . . 64
3.4 FPGA-to-CPU - Remote Side-Channel Attacks on Heterogeneous SoC . . . 66
3.4.1 Introduction . . . 66
3.4.2 Presentation of the Side-Channel Setup . . . 67
3.4.2.1 Side-Channel Targets . . . 67
3.4.2.2 Xilinx Zynq Experimental Setup . . . 67
3.4.3 FPGA-based attack on Hardware AES . . . 68
3.4.4 FPGA-based attack on Software AES . . . 70
3.4.4.1 Experiment 1 : 8-bit Tiny AES . . . 70
3.4.4.2 Experiment 2 : 32-bit OpenSSL AES . . . 71
3.4.5 EM Results & Discussion . . . 73
3.4.5.1 Electromagnetic Side-Channel Attack . . . 73
3.4.5.2 Attack feasibility . . . 74
3.4.5.3 Countermeasures . . . 74
3.4.6 Conclusion . . . 75
3.4.7 Appendix . . . 76
3.5 SCAbox - A Framework for Evaluating the FPGA-based SCA Threat . . . 77
3.5.1 Introduction . . . 77
3.5.1.1 Comparison with Usual Side-Channel Attacks . . . 78
3.5.2 Framework Architecture . . . 79
3.5.2.1 Overview . . . 79
3.5.2.2 Hardware Architecture (FPGA) . . . 79
3.5.2.3 Software Architecture (Processor) . . . 80
3.5.2.4 SCA Automation Tool (Computer) . . . 80
3.5.3 SCAbox User experience . . . 81
3.5.3.1 Use case: SCA Sensor & Setup . . . 81
3.5.3.2 Use case: SCA Acquisition & Attack . . . 82
3.5.3.3 Discussion . . . 82
3.5.4 Conclusion . . . 83

3.6 Conclusion on FPGA-based Power Analysis . . . 84
3.6.1 Results Reminder . . . 84
3.6.1.1 Main Contributions . . . 84
3.6.1.2 Some Numbers . . . 84
3.6.2 *SbHA Knowledge*: Demystifying On-Chip Power SCA . . . 85
3.6.2.1 On-Chip Power SCA can be conducted using digital logic 85
3.6.2.2 On-Chip Power SCA can be conducted with limited sensors . . . 85
3.6.2.3 On-Chip Power SCA can be conducted across the SoC boundaries . . . 85
3.6.3 *SbHA Knowledge*: Strengths and Challenges of On-Chip Power SCA . . . 86
3.6.3.1 Challenge 1: Trace Synchronization . . . 86
3.6.3.2 Challenge 2: Data Storage . . . 86
3.6.3.3 Challenge 3: Data Export . . . 87
3.6.3.4 Strength 1: Sensor Combination . . . 87
3.6.3.5 Strength 2: Unlimited Attack Time . . . 87
3.6.4 A Step Toward SoC Attacks . . . 88

4 Software-based Power Analysis Attacks on Complex SoCs 89
4.1 Chapter Introduction . . . 90
4.1.1 Identifying SbSCA Vectors . . . 90
4.1.2 Selecting an SbSCA Vector . . . 93
4.1.3 Chapter Outline . . . 94
4.2 Technical Background . . . 95
4.2.1 Memory Controller Basics . . . 95
4.2.2 Delay-Line Blocks in Low-Bandwidth Memory Controllers . . . 96
4.2.3 DLLs in High-Bandwidth Memory Controllers . . . 97
4.3 SideLine: Delay-Line-based power SCA on complex SoCs . . . 99
4.3.1 Introduction . . . 99
4.3.2 Experimental Setup . . . 99
4.3.2.1 Tested Devices . . . 99
4.3.2.2 OpenSSL AES Architecture . . . 100
4.3.2.3 Threat Model . . . 100
4.3.3 DLL-based Power Side-Channel Attack . . . 101
4.3.3.1 Validating DLL Effectiveness: *Monitoring Temperature* 102
4.3.3.2 Improving Sampling Rate and Synchronisation using DMA . . . 102
4.3.3.3 Bare Metal OpenSSL AES Attack Setup . . . 103
4.3.3.4 DLL-based SCA Attack on *Zynq-7000* SoC . . . 104
4.3.3.5 Conclusion on DLL-based SCA . . . 106
4.3.4 Delay-Block-based Power Side-Channel Attack . . . 106
4.3.4.1 From Delay-Block to TDC Sensor . . . 106

4.3.4.2 Validating Delay-Block Effectiveness: `strcmp` test . . 107
4.3.4.3 Linux-based OpenSSL AES Attack Setup 108
4.3.4.4 Delay-block-based SCA Attacks on STM32MP1 SoC . 109
4.3.5 Discussion 111
4.3.5.1 Performance and Limitations of *SideLine* 111
4.3.5.2 Hardware & Software Mitigations 112
4.3.6 Conclusion 113
4.4 Additional Results 115
4.4.1 Covert Channels between processes 115
4.4.2 Simple Power Analysis on RSA 117
4.4.2.1 SPA on *square and multiply* RSA version 117
4.4.2.2 SPA on *square and multiply always* RSA version 118
4.4.2.3 SPA on *Montgomery Powering Ladder* RSA version . . 119
4.5 Conclusion on Delay-Line-based Power Analysis 120
4.5.1 Results Reminder 120
4.5.1.1 Main Contributions 120
4.5.1.2 Some Numbers 120
4.5.2 *SbHA Knowledge*: SbHA Attack Vector Detection Methods . . . 121
4.5.2.1 Listing OS user-exposed hardware controls 121
4.5.2.2 Searching registers in the target documentation 121
4.5.2.3 Reversing OS device drivers and boot code 122
4.5.3 From SideLine to FaultLine 124
4.6 Appendix 125

5 Software-based Fault Injection on SoC External Memory Transfers 129
5.1 Chapter Introduction 130
5.2 Technical Background 131
5.2.1 Monitoring Memory Transfers 131
5.2.2 Faulting Memory Transfers 131
5.2.3 Threat Model 132
5.2.4 Persistent Fault Analysis on AES 132
5.3 FaultLine: Software-based Fault Injection on Memory Transfers 136
5.3.1 Experimental Setup and Fault Parameters 136
5.3.1.1 Experimental Setup 136
5.3.1.2 Deeper View of the Fault Mechanism 137
5.3.1.3 Shaping the Glitch 137
5.3.2 *FaultLine* on a Bare-Metal Device 138
5.3.2.1 Characterizing the Injected Faults 139
5.3.2.2 Differential Fault Analysis Attack on AES 140
5.3.2.3 Persistent Fault Analysis Attack on AES 141
5.3.2.4 Conclusion on Bare-Metal Results 143
5.3.3 *FaultLine* on a Device Running Linux 143
5.3.3.1 Attack Setup 143

5.3.3.2 Simple Data Byte Corruption . . . 144
5.3.3.3 Persistent Fault Analysis Attack on AES . . . 145
5.3.3.4 Bellcore Attack on OpenSSL Signatures . . . 145
5.3.3.5 Conclusion on OS Results . . . 146
5.3.4 Discussion . . . 146
5.3.4.1 Related Works . . . 146
5.3.4.2 Advantages and Limitations over Prior Methods . . . 146
5.3.4.3 Countermeasures . . . 147
5.4 Conclusion on Delay-Line based Fault Injection . . . 149
5.4.1 Results Reminder . . . 149
5.4.1.1 Main Contributions . . . 149
5.4.1.2 Some Numbers . . . 149
5.4.2 *SbHA Knowledge*: Toward Large Scale SbHAs . . . 150

6 Conclusion and Perspectives **153**
6.1 Manuscript Summary . . . 154
6.2 Conclusion on SbHA . . . 155
6.3 Thesis Impact . . . 156
6.4 SbHA Perspectives . . . 157
Bibliography . . . 159

LIST OF FIGURES

1 Classification des attaques matérielles à distance xxvii
2 Étude des attaques SbSCAs à base de FPGA xxix
3 Les trois scénarios d'attaques de *SideLine* xxxiii
4 De *SideLine* à *FaultLine* xxxv

1.1 Thesis Roadmap 4

2.1 Proposed hardware attack classification 9
2.2 Local Non-Invasive Attack Setup 10
2.3 Simple Power Analysis on RSA 12
2.4 Evolution of SIM 22
2.5 SGX and TZ-based Trusted Execution Environments 23
2.6 Artist's representation of the TEMPEST attack threat 24
2.7 Famous exploit names in the microarchitectural attack field 26
2.8 Local hardware attack versus software-based hardware attack 28
2.9 Artist's representation of the Rowhammer vulnerability 30
2.10 Cloud FPGA fabrics shared between multiple users 33
2.11 FPGA-based power side-channel attack threat 38
2.12 SbHA paths and privilege levels 43
2.13 Proposed attack classification 45

3.1 Chapter contributions 48
3.2 RLC parasitic elements in the PDN 50
3.3 Inverter gate propagation delay 51
3.4 Functional schematic of a TDC-based sensor 52
3.5 Functional schematic of a RO-based sensor 53
3.6 Overview of FPGA-based Power Side-Channel Exploits 54
3.7 FPGA-to-FPGA attack 55
3.8 Schematic of the proposed JRO-based sensor design 57
3.9 Effect of the number of JRO-based sensors on the overall resolution . . . 58
3.10 Xilinx Zynq Multi-User Experimental Setup. 60
3.11 CPA Attack Results on AES 61
3.12 CPA attack results using various sensing mechanisms 63

3.13 FPGA-to-CPU attack . . . 66
3.14 Xilinx Zynq experimental side-channel setup . . . 67
3.15 Averaged AES power supply fluctuation . . . 68
3.16 Logical distance between sensors and target algorithm. . . . 69
3.17 Power supply fluctuation resulting from 100 Tiny AES encryptions . . . 70
3.18 Power supply fluctuation resulting from 100 OpenSSL AES encryptions . 71
3.19 Correlation rate over the time . . . 72
3.20 XRAY picture and EM setup . . . 73
3.21 Logic schematic and implemented design of one TDC-based sensor instance 76
3.22 SCAbox Logo . . . 77
3.23 Reducing SCA setup complexity . . . 78
3.24 SCAbox hardware (VHDL) and software (C) architecture . . . 80
3.25 SCAbox CPA Results . . . 81
3.26 From FPGA to SoC SbHA exploits . . . 88

4.1 Benchmarking the resources available for voltage sensing . . . 91
4.2 STM32MP1 *Delay Block* Diagram taken from [93] . . . 92
4.3 Typical SoC connectivity with external memories . . . 95
4.4 An example of DLB used in low-bandwidth memory controllers. . . . 96
4.5 An example of DLL used in DDR memory controllers. . . . 97
4.6 Three core-vs-core attack variants . . . 101
4.7 DLL response to sudden temperature drops . . . 102
4.8 DLL-based attack results . . . 105
4.9 Effect of on-chip voltage variations on the sampled delay values. . . . 107
4.10 Sampled delay values displayed on screen. . . . 107
4.11 `strcmp` test . . . 108
4.12 *SideLine* CPA demonstration . . . 109
4.13 AP-vs-MCU attack results . . . 110
4.14 MCU-vs-AP attack results . . . 111
4.15 Power fluctuation-based covert-channel captured using delay-lines . . . 115
4.16 *Square and multiply* RSA delay-line-based SPA . . . 117
4.17 *Square and multiply always* RSA delay-line-based SPA . . . 118
4.18 *Montgomery powering ladder* RSA delay-line-based SPA . . . 119
4.19 From *SideLine* SbSCA to *FaultLine* SbFIA Exploits . . . 124
4.20 *Zynq-7000* AP-vs-AP attack scenario . . . 125
4.21 *STM32MP1* AP-vs-MCU attack scenario . . . 126
4.22 *STM32MP1* MCU-vs-AP attack scenario . . . 127
4.23 *STM32MP1* MCU-vs-AP attack . . . 128

5.1 Memory transfer organization between a SoC and an external SDRAM . . 131
5.2 PFA ciphertext byte 0 distribution 134
5.3 Method for extracting the right PFA key candidate 135
5.4 Nominal vs faulty strobe phase-shifts 137
5.5 `stress` and `width` characterization 140
5.6 Progression of the PFA distribution over 17,053 faulty ciphertexts 142

LIST OF TABLES

2.1 The impact of 40 years of research in the semiconductor industry 19
2.2 Non-Exhaustive list of Rowhammer exploits 31
2.3 List of FPGA-based Power Glitch Injection exploits 34
2.4 List of DVFS-based power glitch injection exploits 35
2.5 List of delay-line-based glitch injection exploits 36
2.6 List of FPGA-based power SCA exploits 39
2.7 List of ADC-based Power SCA exploits 40
2.8 List of Delay-line-based Power SCA exploits 41
2.9 List of RAPL-based power SCA exploits 41

3.1 Resource utilization for 1 RO-based sensor instance. 57
3.2 TDC optimization impact 69
3.3 Number of traces required to retrieve an AES key byte 73

4.1 On-chip SCA vector benchmark results. 93
4.2 Overall delay-line-based power SCA results. 112

5.1 Variations in faulty ciphertexts 141

GLOSSARY

ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AI	Artificial Intelligence
AP	Application Processor
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
CEMA	Correlation ElectroMagnetic Analysis
CPA	Correlation Power Analysis
CPU	Computer Processing Unit
CRT	Chinese Remainder Theorem
DDR	Double Data Rate
DFA	Differential Fault Analysis
DoS	Denial-of-Service
DLB	Delay-Line Block
DLL	Delay-Locked-Loop
DMA	Direct Memory Access
DPA	Differential Power Analysis
DRAM	Dynamic Random-Access Memory
DRM	Digital Right Management
DUT	Device Under Test
DVFS	Dynamic Voltage and Frequency Scaling
ECC	Error-Correction Code
EM	ElectroMagnetic
eSIM	embedded SIM
FIA	Fault Injection Attack
FIFO	First In First Out
FPGA	Field-Programmable Gate Array

GPU	Graphic Processing Unit
HPC	Hardware Performance Counters
IC	Integrated Circuit
IoT	Internet of Things
IP	Intellectual Property
iSIM	integrated SIM
JRC	Johnson Ring-Counter
JRO	Johnson Ring-Oscillator
MCU	Microcontroller Unit
MMU	Memory Management Unit
NSA	National Security Agency
NVM	Non-Volatile Memory
OS	Operating System
PCB	Printed Circuit Board
PDN	Power Delivery Network
PIN	Personal Identification Number
PFA	Persistent Fault Attack
PHY	Physical Layer
PVT	Process Voltage Temperature
RAM	Random-Access-Memory
RAPL	Running Average Power Limit
RHA	Remote Hardware Attack
RNG	Random Number Generator
RO	Ring-Oscillator
RSA	Rivest–Shamir–Adleman
SbHA	Software-based Hardware Attack
SbSCA	Software-based Side-Channel Analysis
SbFIA	Software-based Fault Injection Attack
SCA	Side-Channel Analysis
SDRAM	Synchronous Dynamic Random-Access Memory
SEV	Secure Encrypted Virtualization
SIM	Subscriber Identity Module
SGX	Software Guard Extensions

SoC	System-on-Chip
SPA	Simple Power Analysis
TDC	Time-to-Digital Converter
TEE	Trusted Execution Environment
TEMPEST	Telecommunications Electronic Material Protected from Emanating Spurious Transmissions
TZ	TrustZone
VCO	Voltage-Controlled Oscillator
VM	Virtual Machine
VRM	Voltage Regulator Module

AVANT PROPOS

C'est à la fin de mon stage de fin d'étude dans l'entreprise Gemalto que Philippe Loubet Moundi m'a proposé une place en tant que doctorant dans l'équipe Hardware Lab. Pourtant, la période troublée que traversait Gemalto durant le rachat par Thales n'était pas forcément propice à ce recrutement. Philippe m'a fait confiance et je lui en suis reconnaissant.

Jean-Max Dutertre mon directeur de thèse à l'école des Mines de Saint-Étienne et Yannick Teglia mon encadrant Thales ont su par leurs conseils avisés et leurs observations habiles me mettre directement sur de bons rails. Une thèse c'est comme une maison, sans fondations solides ça s'effondre. J'ai eu la chance d'avoir un encadrement en béton armé.

Je voudrais remercier chaleureusement toutes les personnes que j'ai pu rencontrer durant de cette thèse. Ma pensée va à mes collègues du Centre Microélectronique de Provence à Gardanne et à l'équipe Security Lab de Thales à La Ciotat. Merci pour votre accueil et pour tous les échanges productifs (ainsi que non-productifs) que nous avons eu.

Je remercie ma famille Elina, Maman, Papa, Manon, Rémi, Adèle, Ugo, Lila et Robin pour leur soutien sans faille malgré la distance qui nous sépare.

Finalement je remercie les examinateurs Jacques Fournier, Lilian Bossuet, Emmanuel Prouff et les rapporteurs Edith Kussener et Régis Leveugle qui ont accepté de se livrer au jeu de la relecture et de la soutenance. Leurs questions et leurs remarques constructives ont contribué à l'aboutissement de ce manuscrit.

RÉSUMÉ

Introduction

Durant l'année 1958, Jack Kilby et son équipe élaboraient le premier circuit intégré à transistor dans les laboratoires de Texas Instrument. À peine onze ans plus tard, les premiers ordinateurs de bord composés de plusieurs milliers de transistors nous amenaient sur la Lune. Depuis lors, l'industrie du semi-conducteur a conquis l'intégralité des aspects de la vie humaine : communication, travail, transports, etc. La loi de Moore prévoyant un doublement annuel du nombre de transistors dans les circuits intégrés a été satisfaite pendant 55 années consécutives et nos ordinateurs personnel sont maintenant composés de milliards de transistors.

Si la découverte des circuits intégrés a été une révolution instantanée, leur besoin en sécurité n'est devenu évident que 30 ans plus tard avec l'apparition de services aux consommateurs tels que : les ordinateurs personnels, la téléphonie mobile et l'avènement d'Internet. Pendant une longue période cependant, la structure interne des circuits intégrés a été considérée comme inviolable et la plupart des vulnérabilités ont été recherchées au niveau logiciel.

Les travaux menés par Van Eck [36], Biham [13] et Kocher [77] durant les années 80-90 ont prouvé que ce postulat était faux. Ces chercheurs ont participé à introduire de nouvelles techniques d'attaques telles que l'étude des canaux cachés, ou Side-Channel Analysis (SCA) en anglais, et l'injection de faute, ou Fault Injection Attack (FIA) en anglais, qui visent à exploiter des vulnérabilités physiques liées aux ressources matérielles qui composent les circuits intégrés.

Depuis plus de 30 ans, l'étude de la sécurité matérielle des circuits intégrés s'est montrée nécessaire afin d'implémenter des systèmes résistants aux attaques menées par des groupes de pirates informatiques. Les attaques matérielles utilisent des équipements tels que des lasers, des injecteurs électromagnétiques et des oscilloscopes pour injecter des erreurs dans les calculs d'un circuit ou pour espionner son fonctionnement. L'objectif à terme est d'en extraire les secrets comme par exemple le code source pour voler la propriété intellectuelle, les clés cryptographiques ou les données utilisateurs afin de les vendre au plus offrant. Les attaques matérielles visent une grande variété d'appareils comme les cartes à puce (cartes SIM, cartes bancaires, passeports), les processeurs (téléphones, ordinateurs, serveurs) et tous les autres objets pour lesquels l'extraction d'information fait sens (microcontrôleurs pour l'IoT, automobiles, consoles de jeu, drones, etc.).

Les attaques matérielles sont globalement utilisées sur des systèmes légers qui peuvent être volés et placés sur un banc d'attaque. À la différence des attaques logicielles,

elles nécessitent l'accès physique à la cible et ne permettent pas d'attaques à grande échelle sur des milliers d'appareils en simultané. Cependant, elles constituent une menace réelle et restent très étudiées car elles permettent, dans de nombreux cas, de révéler des secrets qu'une attaque logicielle ne pourrait pas obtenir. L'extraction de données secrètes contenues dans un appareil électronique peut avoir de grandes répercussions sur la crédibilité de gouvernements, sur la réputation et la rentabilité d'entreprises ainsi que sur la vie privée des consommateurs.

Récemment, plusieurs études menées sur des processeurs complexes ont montré que les attaques matérielles n'étaient plus nécessairement locales. Avec la multiplication et la complexification des objets connectés, il est devenu possible de mener ces attaques à distance.

Cette thèse propose d'étudier une nouvelle famille d'attaque communément nommée "attaque matérielle à base de logiciel", ou Software-based Hardware Attack (SbHA) en anglais, qui à travers l'exécution d'un programme malicieux vise à utiliser des composants intégrés dans un circuit cible afin de mener des attaques matérielles. Ces attaques ne requièrent aucun équipement externe à l'appareil ciblé et peuvent donc être menées à distance.

Le but de ce travail est tout d'abord d'étudier les risques découlant des attaques matérielles à distance, notamment au regard de l'adoption massive de la sécurité intégrée. Mais aussi d'évaluer de nouvelles vulnérabilités afin de mieux cerner l'étendue de la menace. Cette thèse contient des travaux variés allant d'attaques SCA embarquées sur des circuits intégrés reprogrammables de type Field-Programmable Gate Array (FPGA) à de l'attaque FIA sur des processeurs complexes de type System-on-Chip (SoC). Tous ces travaux ont été menés de façon à être reproductibles et réalisables à distance. Ils visent à préparer les différents acteurs de la sécurité intégrée à cette menace naissante et ainsi à éviter sa mise en exécution future sur des milliers d'appareils connectés à travers le monde.

Dans la suite de ce résumé, nous décrivons les grandes lignes de chaque chapitre composant ce manuscrit. Le chapitre 2 présente le sujet et l'état de l'art des attaques matérielles à distance. Les chapitres 3, 4 et 5 décrivent les expérimentations réalisées durant cette thèse. Finalement, le dernier chapitre conclut sur les résultats obtenus et revient sur l'impact des travaux menés et leurs perspectives futures.

Les attaques matérielles à distance

Le chapitre 2 de ce manuscrit est dédié à la description du sujet de cette thèse : "les attaques matérielles à distance des objets connectés" et à la classification des travaux existants. Dans un premier temps, nous décrivons l'origine des attaques matérielles (Section 2.1.1) et procédons à leur répartition en 3 familles (Section 2.1.2) : les attaques invasives, les attaques semi-invasives et les attaques non-invasives comme proposé dans [125]. Il apparait que les attaques matérielles à distance ne s'intègrent dans aucune de ces familles qui nécessitent toutes des équipements et un accès physique à la cible. Nous proposons

donc d'en ajouter une appelée "attaque matérielle à base de logiciel" qui permettrait de classer ce nouveau type d'attaque.

La suite du chapitre présente l'origine et les enjeux derrière la multiplication des objets connectés. Dans la section 2.2, nous décrivons les mécanismes qui ont mené à l'adoption d'objets connectés dans tous les aspects de l'activité humaine (industrie, services publics, cloud, etc). Puis, nous décrivons les dangers qui découlent de l'utilisation de ces systèmes et comment des pirates tirent parti de certaines avancées pour extorquer, menacer et soudoyer des états, des entreprises ou des utilisateurs. Enfin, nous évoquons le futur des attaques matérielles dans la sous-section 2.2.3 et imaginons comment la sécurité matérielle évoluera face aux mutations des circuit intégrés et à la multiplication de services de calcul distant comme le cloud qui ne permettent pas de lancer des attaques matérielles locales classiques car inaccessibles. L'hypothèse de résistance aux attaques matérielles qu'apportent ces services de calcul distant est cependant remise en cause par l'introduction d'attaques matérielles à distance.

Dans la section 2.3, nous décrivons en quoi consiste une attaque matérielle à distance. Nous identifions tout d'abord trois mécanismes qui favorisent la multiplication de ces attaques. Le premier est lié à la complexification des circuits modernes. Les circuits de type SoC sont composés d'un processeur et de multiples ressources matérielles (mémoire, capteurs, régulateurs) sur la même puce de silicium. En rassemblant ces diverses fonctionnalités sur un même système, les concepteurs améliorent grandement les capacités de calcul et les performances énergétiques du circuit. Les ressources matérielles peuvent maintenant être accédées et programmées directement par un logiciel pour adapter par exemple la consommation du circuit à l'activité de l'utilisateur. La coopération matérielle-logicielle est plus que jamais représentée et permet d'obtenir des performances inégalées. Cependant, malgré les bénéfices, l'accès à des ressources matérielles par le logiciel pourrait être l'origine de nouvelles attaques. Notamment les attaques matérielles à distance.

Le second mécanisme qui favorise l'émergence des attaques matérielles à distance est la démocratisation de systèmes comprenant plusieurs utilisateurs et/ou plusieurs niveaux de privilèges. Le partage de ressources matérielles entre plusieurs utilisateurs est devenu la référence avec l'avènement du cloud. L'isolation entre les processus est assurée par des mécanismes de protection mémoire et des niveaux de privilèges offerts par les systèmes d'exploitation, ou Operating System (OS) en anglais. Le fait de rassembler plusieurs utilisateurs avec différent niveaux de privilège sur un même système ouvre la porte à de nouvelles attaques où un utilisateur tente d'outrepasser l'isolation en place pour : espionner les autres utilisateurs, procéder à de l'escalade de privilège ou générer un déni de service. Les OS sont devenus si complexes qu'il en devient difficile d'en garantir la fiabilité sécuritaire. Pour cette raison, il est désormais commun d'intégrer des entités de sécurité matérielle afin d'assurer un niveau de sécurité supérieur dans les produits contenant des informations sensibles. L'intégration d'entités de sécurité prétendues résistantes aux attaques logicielles est le troisième mécanisme qui pourrait favoriser l'émergence d'attaques matérielles à distance.

Depuis les années 80, des cartes à puce telles que les cartes SIM assurent la sécurité des appareils connectés. Aujourd'hui des circuits intégrés effectuant les mêmes opérations sécuritaires mais pouvant être directement soudés sur un carte mère (eSIM) ou intégrés à l'intérieur d'un SoC (iSIM) remplacent peu-à-peu le format carte. Tant que les circuits sécuritaires étaient localisés en dehors du SoC, ils pouvaient difficilement être attaqués par un processus malicieux s'exécutant dans le processeur. Cependant, l'intégration de ces entités sécuritaires sur la même puce de silicium que le processeur applicatif ouvre la voie à de nouvelles vulnérabilités internes provenant du matériel. Ainsi, nous verrons que le risque de voir ces entités soumises à des attaques matérielles à distance augmente.

Un autre aspect de la sécurité intégrée est l'avènement des plateformes d'exécution sécurisées, ou Trusted Execution Environment (TEE) en anglais, renforcées par des entités matérielles conçues par des fabricants de processeurs majeurs tels que ARM, INTEL et AMD. Ces plateformes appelées TrustZone (TZ), Software Guard Extensions (SGX) et Secure Encrypted Virtualization (SEV) partent du principe que l'OS implémenté sur le processeur applicatif n'est pas forcément fiable et qu'il faut déléguer les opérations sécuritaire à une entité indépendante de l'OS. Ces entités assurent donc les opérations critiques telles que l'authentification ou le chiffrement à la place de l'OS et ajoutent une nouvelle couche de sécurité au-delà des privilèges administrateurs. Les attaques matérielles à distance pourrait préférentiellement s'attaquer à ces plateformes qui n'ajoutent que de la sécurité du point de vue logiciel et qui restent donc le plus souvent vulnérables aux attaques physiques.

Sur les facteurs favorisants les attaques matérielles à distance on retiendra : la complexification des circuits intégrés, l'avènement des systèmes multi-tâches/multi-utilisateurs à plusieurs niveaux de privilèges et l'intégration d'entités de sécurité dans les SoCs modernes.

Dans la suite du chapitre 2, nous classons les attaques matérielles à distance en trois familles distinctes dont l'une d'elle sera le sujet d'étude de cette thèse. Ces trois familles ont pour point commun de rendre les attaques à distance possibles mais elles n'utilisent pas les mêmes moyens. Ces attaques sont classées dans la figure 1, nous les identifions dans la liste ci-dessous :

- *Les attaques TEMPEST*. L'étude des attaques TEMPEST remonte à la guerre froide. C'est un nom de code de la National Security Agency (NSA) [101] faisant référence à l'extraction d'informations contenues dans des appareils électroniques à travers des émanations physiques pouvant être mesurées à longue portée. Ce sont par exemple des émanations électromagnétiques générées par un appareil et pouvant être capturées à plusieurs mètres en utilisant une antenne radio. Elles avaient été introduites au grand public en 1985 dans un article de Wim van Eck [36] présentant l'extraction de l'image affichée par un écran d'ordinateur situé à l'intérieur d'un bâtiment à partir d'une antenne embarquée dans une camionette postée à l'extérieur (à une distance de 10 mètres).

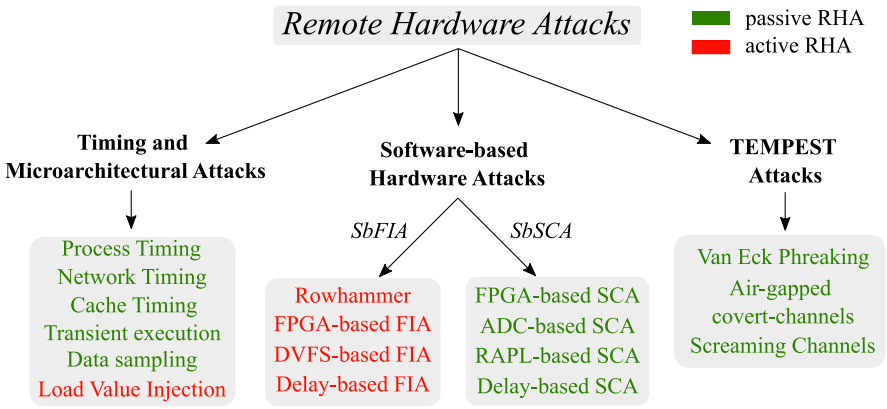


Figure 1 Classification des attaques matérielles à distance comme proposé dans le chapitre 2

Ces attaques ont récemment été remises au-devant de la scène par l'article *Screaming Channel* [22, 21] publié en 2018 qui vise des processeurs hétérogènes composés de logique analogique et digitale. Ces recherches ont prouvé que l'activité digitale génératrice de fuites par canaux-cachés pouvait se retrouver amplifiée par les modules intégrés analogiques de radio-fréquence (Wifi, Bluetooth) puis capturée à distance en utilisant une radio logicielle (SDR). En plus de capter l'activité électromagnétique induite par la cible, cet article va plus loin en démontrant la possibilité d'extraire des informations sur les processus exécutés par l'appareil espionné. Les chercheurs démontrent que l'empreinte side-channel d'un processus cryptographique s'exécutant dans la cible est détectable dans le signal radio et mènent une SCA permettant l'extraction de la clé secrète (à 15 mètres de la cible). Avec la prolifération des objets connectés qui intègrent de plus en plus souvent des modules radio, le champ d'application de ces attaques devrait s'élargir dans le futur. L'avantage ici pour l'attaquant est qu'il n'y plus de besoin d'accéder directement à la cible pour effectuer la SCA. Pour cette raison, nous considérons ces travaux comme des attaques matérielles à distance.

- *Les attaques microarchitecturales et temporelles.* Les attaques temporelles sur les processus cryptographiques et les protocoles réseaux sont connues depuis les travaux menés par Kocher [77] en 1996. Ces attaques peuvent être réalisées à distance puisqu'elles nécessitent seulement la mesure d'un temps d'exécution pour l'extraction des secrets. Aujourd'hui la plupart des processus cryptographiques sont protégés contre ce type d'attaque et utilisent des algorithmes à temps constant pour éviter toute fuite d'information liée à une clé cryptographique ou à un mot de passe.

  Depuis le milieu des années 2000, de nouvelles attaques exploitant des vulnérabilités temporelles ont fait leur apparition : les attaques microarchitecturales. Ces menaces tirent parti du partage de certaines ressources matérielles entre plusieurs processus et utilisateurs pour mettre à mal l'isolation mémoire implémentée dans

ces systèmes. Par exemple, les attaques temporelles sur la mémoire cache exploitent le fait que des données provenant d'une application peuvent être stockées de manière transitoire dans la mémoire cache. En mesurant les temps d'accès à la mémoire, des processus espions peuvent déduire l'activité de programmes victimes et même établir des communications cachées entre processus [144, 58, 84]. Ces attaques sont considérées comme des attaques physiques car elles exploitent des vulnérabilités provenant du matériel et non de l'implémentation logicielle des programmes.

Plus récemment des attaques telles que *Spectre* [75] et *Meltdown* [86] ont visé des optimisations matérielles des processeurs telles que la prédiction de branchement ou l'exécution spéculative afin d'accéder à des données protégées.

Les attaques microarchitecturales sont à la frontière entre des attaques logicielles et des attaques matérielles. Elles sont menées à travers un programme malicieux mais exploitent des vulnérabilités matérielles. Elles s'inscrivent donc dans la liste des attaques matérielles à distance mais n'utilisent pas les mêmes outils et n'exploitent pas les mêmes vulnérabilités que les attaques matérielles locales.

- *Les attaques matérielles à base de logiciel* (SbHA). Ces attaques sont le cas d'étude de cette thèse. Elles sont des reproductions pures des attaques matérielles classiques menées dans des laboratoires, à l'exception du fait qu'elles peuvent être réalisées à distance. C'est à dire qu'elles cherchent à induire des fautes ou à collecter une fuite side-channel de la même façon qu'une attaque matérielle classique mais sans utiliser d'équipement.

  Les SbHA tirent parti de ressources matérielles directement implémentées dans les circuits cibles afin de mener des attaques. Elles profitent de la complexité des SoCs qui comportent des dizaines de composants matériels et dont l'accès et la calibration peut permettre l'implémentation d'attaques physiques. Dans la section 2.3.4 et 2.3.5 nous dressons un état de l'art complet de ces attaques qu'elles soient dédiées à la FIA ou à la SCA.

  Du point de vue de la FIA, nous définissons les attaques réalisées sous le nom d'*injection de faute à base de logiciel*, ou Software-based Fault Injection Attack (SbFIA) en anglais. Les familles d'attaques décrites comportent la SbFIA sur mémoire DRAM ou *Rowhammer* introduite en 2014 par Kim et al [74], la FIA à base de régulateur tension-fréquence avec *ClkSCREW* [128], *VoltJockey* [111] et *Plundervolt* [100], les attaques sur FPGA telles que *FPGAhammer* [80] et enfin l'attaque *FaultLine* [51] découverte lors de cette thèse et décrite dans le chapitre 5.

  Du point de vue de la SCA nous appelons ces attaques *analyse par canaux cachés à base de logiciel*, ou Software-based Side-Channel Analysis (SbSCA) en anglais. Nous intégrons plusieurs types d'attaques à la classification : les attaques SCA à base de capteurs FPGA [122] qui seront étudiées dans le chapitre 3 de ce manuscrit, les attaques à base de convertisseurs analogiques-numériques telles que *Leaky*

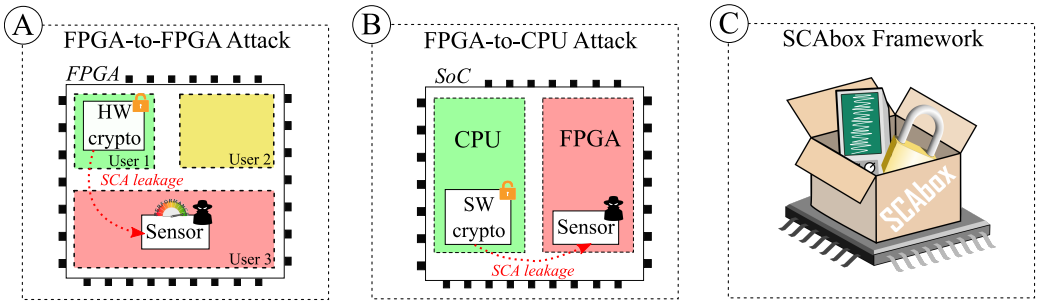


Figure 2 Étude des attaques SbSCAs à base de FPGA : Contributions du chapitre 3

Noise [46], les attaques à base de lignes à retard (delay-lines) menées dans cette thèse qui seront décrites au chapitre 4 (*SideLine* [51]) et enfin les attaques SCA menées sur des processeurs Intel dans l'article *Platypus* [85].

La section 2.4 conclut ce chapitre en proposant une vue globale des trois familles d'attaques matérielles à distance identifiées et des différents groupes de vecteurs d'attaques contenus dans chacune d'elles. La classification globale est illustrée dans la figure 1 de ce résumé.

Attaques par analyse de consommation à base de FPGA

En 2018, au début de cette thèse, plusieurs types de mécanismes permettant les SbHA tels que Rowhammer [74] ou les régulateurs tension/fréquence [128] avaient déjà été identifiés. Ces travaux précédents étaient, en grande majorité, dédiés à des attaques SbFIA. Seuls les travaux réalisés par Schellenberg et al. sur FPGA présentaient la menace des attaques SbSCA [122].

Il apparut rapidement que l'étude de ces travaux sur FPGA pourraient être un point de départ intéressant pour cette thèse. En étudiant des FPGAs qui permettent l'implémentation de n'importe quel type de bloc numérique, un banc d'expérimentation complet contenant un capteur de consommation intégré, la cible (un algorithme cryptographique) et le stockage des données pour les courbes SCA peut être mis en place. De plus avec l'adoption des FPGAs dans les SoC modernes [4] ainsi que dans les services cloud [24], le risque d'attaque lié à ces entités devient critique [131] et suscite l'intérêt de la recherche. L'implémentation de FPGA partagés entre plusieurs utilisateurs est déjà possible [24] et elle pourrait faire son apparition dans le cloud. Il y a donc un véritable enjeu sécuritaire à étudier ce type d'appareils au-delà de l'étude de faisabilité des attaques SbSCA.

La figure 2 illustre trois travaux décrits dans le chapitre 3 de ce manuscrit. Tout d'abord, dans la section 3.3 nous étudions des capteurs de délai à base de Ring-Oscillators (ROs) permettant de capturer les variations de consommation électrique dans les FPGAs et menons des attaques SCA entre blocs logiques. Cette première partie étudie donc les attaques sur FPGA multi-utilisateurs où l'un des clients utilise sa logique reprogrammable pour espionner l'activité d'autres blocs logiques appartenant à d'autres clients (attaques

FPGA-to-FPGA dans la figure 2.A). À partir de capteurs digitaux optimisés pour la SCA nous lançons des attaques de type Correlation Power Analysis (CPA) [18] sur des cibles cryptographiques telles qu'un module Advanced Encryption Standard (AES) matériel [30]. Ainsi nous démontrons que la fuite de consommation liée à l'activité d'un module AES (implémenté dans une partie du FPGA) peut être collecté par des capteurs digitaux (implémentés dans une autre partie de FPGA). Cette étude confirme l'hypothèse que les variations de consommation électrique liées à l'activité des transistors se propagent dans toute la puce de silicium est peuvent être mesurées par une entité espacée physiquement de l'origine de la fuite. Il est donc possible de mener des attaques SbSCA entre des blocs matériels formant un FPGA ou un SoC (si tant est que l'attaquant trouve un moyen de collecter la fuite de consommation).

Une deuxième conclusion de cette étude est que les limitations en terme de fréquence des capteurs intégrés (ici 250 MHz) ne sont pas forcément rédhibitoires pour la mesure de la fuite de consommation. En effet, à travers nos expérimentations, nous remarquons que la fuite side-channel reste exploitable malgré l'augmentation de la vitesse d'exécution du module victime. Nous réalisons plusieurs tests sur le module AES en modifiant sa fréquence de fonctionnement de 10 MHz à 200 MHz et n'observons pas de réelle différence sur les corrélations relevées par la CPA. Ces résultats viennent confirmer que la fuite side-channel est essentiellement liée à la structure d'alimentation du circuit à sa finesse de gravure [31, 59] et non à la fréquence d'horloge du module ciblé [90]. Il sera donc possible d'attaquer des systèmes très rapides tels que des processeurs complexes cadencés au GHz avec des capteurs lents de l'ordre du MHz comme en témoignent les résultats du chapitre 4. Il est cependant important de prendre en compte qu'une trop grande limitation de la fréquence d'échantillonnage pourra s'avérer problématique pour la détection des points d'intérêts et ainsi la resynchronisation des courbes collectées.

La section 3.4 du chapitre 3 étudie des systèmes hétérogènes composés à la fois d'un FPGA et d'un processeur (Attaques FPGA-to-Computer Processing Unit (CPU) dans la figure 2.B). L'objectif de cette section est de démontrer que des attaques SbSCA sont possibles d'un composant du SoC vers un autre. Ici, un SoC Xilinx Zynq 7000 [141] composé d'un processeur ARM Cortex-A et d'un FPGA est adopté pour mener des attaques du FPGA vers le CPU. C'est-à-dire, la collecte d'une fuite de consommation électrique induite par un algorithme cryptographique logiciel s'exécutant sur le CPU par des capteurs implémentés dans le FPGA. À travers l'implémentation de capteurs de type Time-to-Digital Converter (TDC) [151] dans le FPGA et la mise en place de plusieurs implémentations d'AES logiciels dans le CPU (OpenSSL AES [106] et Tiny AES [78]), nous lançons des attaques CPA du FPGA vers le CPU. Les résultats obtenus montrent que la fuite side-channel induite par le fonctionnement des AES s'exécutant dans le CPU est suffisamment importante pour être capturée par les capteurs et permet l'extraction des clés cryptographiques. Malgré la distance physique entre les capteurs (dans le FPGA) et l'algorithme (dans le CPU), le fait que les deux entités sont implémentées sur la même puce de silicium établit une canal d'information suffisant pour faire fonctionner l'attaque.

Les attaques du FPGA vers le CPU nécessitent la collecte d'un plus grand nombre de courbes SCA (autour de 100,000 courbes) que les attaques FPGA vers FPGA (autour de 1,500 courbes) mais restent largement envisageables (moins de 15 minutes pour récupérer la clé secrète de l'AES OpenSSL dans notre preuve de concept).

Afin de comparer l'efficacité des attaques intégrées avec les techniques traditionnelles, une SCA électromagnétique est réalisée sur les mêmes AES cibles dans la section 3.4.5. Les résultats obtenus montrent que les capteurs FPGAs sont aussi efficaces que l'analyse électromagnétique malgré les performances limitées des capteurs en terme de vitesse d'échantillonnage et de résolution. L'avantage des TDCs dans cette configuration est leur proximité à la cible qui permet d'éviter toute perte d'information par effet de dissipation de la fuite à travers le package et de limiter l'effet de filtrage induit par les condensateurs de découplage externes. Il semble probable que dans certains SoC complexes comportant des mémoires empilées, de multiples cœurs et de multiple mécanismes de régulation de tension, l'analyse interne de la consommation pourrait être plus performante à terme que l'analyse externe.

La section 3.5 conclut la partie expérimentation du chapitre 3 en présentant un outil open-source permettant la mise en place d'attaques FPGA-vers-FPGA et FPGA-vers-CPU présentées dans les sections précédentes. Ce projet appelé *SCAbox* est disponible sur GitHub et vise à faciliter l'implémentation et l'évaluation de ces attaques [52]. Il intègre plusieurs modules de capteurs FPGA et fournit une méthode simple pour ajouter des algorithmes victimes et les évaluer. Cet outil est purement dédié à la recherche et se veut être utilisé comme plateforme d'initiation aux attaques matérielles à distance afin de mieux appréhender leur menace.

La conclusion du chapitre 3 revient sur les résultat des travaux menés sur FPGA qui ont donné lieu à la publication de deux articles de conférence pour les travaux d'attaque FPGA-vers-FPGA [50] et FPGA-vers-CPU [54] ainsi que la publication en ligne de l'outil *SCAbox* [52].

Nous revenons aussi sur l'impact des résultats obtenus pour l'avancée de cette thèse. L'objectif de celle-ci étant à terme de mener des attaques sur n'importe quel type de SoC implémentant ou non un FPGA. Au moment de clore ce chapitre, il semble désormais acquis que les attaques SbSCA peuvent être menées à partir de capteurs digitaux directement implémentés dans un SoC. Que les attaques SbSCA sont faisables même avec des capteurs moins performants que ceux utilisés pour les attaques matérielles locales. Enfin, que les attaque SbSCA brisent les frontières logicielles qui séparent normalement les différents blocs matériels formant un SoC (c'est à dire toutes les entités matérielles implémentées sur une même puce de silicium). Ceci ouvre la voie à des attaques FPGA-vers-CPU mais aussi à des attaques provenant de n'importe quel capteur (FPGA, Analog-to-Digital Converter (ADC), delay-line) vers n'importe quelle entité du SoC (CPU, Microcontroller Unit (MCU), élément sécurisé, iSIM, etc).

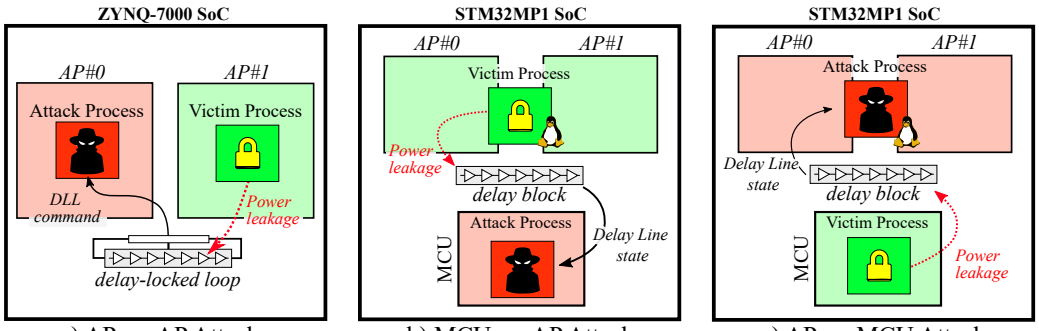
Attaques par analyse de consommation dans les SoC complexes

Le chapitre 4 de ce manuscrit se base sur les enseignements et les conclusions tirés du chapitre 3 pour étendre les attaques SbSCA à des SoC complexes. À la différence des attaques menées précédemment, les travaux réalisés dans ce nouveau chapitre ne nécessitent pas l'utilisation d'un FPGA. En effet, une très faible partie des SoCs embarque un FPGA et il est donc nécessaire d'identifier une entité plus globale pour démontrer la généricité des SbHA. Pour ce faire, nous cherchons dans la section 4.1 les ressources matérielles à disposition dans une vaste majorité de SoCs pour lancer des attaques SbSCA.

La section 4.1.1 présente la méthode mise en place pour identifier de tels vecteurs d'attaque. Plusieurs fiches techniques de SoCs provenant de divers fabricants sont analysées et nous identifions plusieurs modules tels que des ADCs ou des capteurs intégrés qui pourraient être utilisés pour collecter l'activité électrique du circuit ciblé. À travers cette recherche, des modules décris sous le nom de Delay-Line Block (DLB) et Delay-Locked-Loop (DLL) ont été identifiés dans chacun des systèmes testés. Ces composants largement implémentés dans les contrôleurs dédiés à de la mémoire externe sont là pour assurer un déphasage constant entre les signaux d'horloges et de données lors d'accès mémoires réalisés par le SoC. Les DLBs et DLLs sont conçus pour contrebalancer les variations de tension et de température qui ont un impact non-négligeable sur la vitesse de propagation des signaux logiques et qui pourraient induire des erreurs de transfert mémoire. Il y a donc une forte relation entre ces modules DLBs et DLLs et la consommation électrique du système. Étant donné leur structure basée sur des delay-lines, ces blocs sont très proches des capteurs TDCs utilisés pour collecter la fuite de consommation dans les FPGAs. Nous avons donc rapidement soupçonné que ces composants pourraient s'avérer efficaces pour la capture de l'activité électrique d'un système cible et devenir des vecteurs d'attaques SbSCA.

Pour des raisons de vitesse d'échantillonnage, de facilité d'accès et de généricité, il a été décidé dans la section 4.1.2 d'étudier les DLBs et les DLLs plutôt que le reste des composants disponibles dans les SoCs étudiés. La section 4.3 présente l'attaque *SideLine* et les expérimentations menées dans deux SoCs : le STM32MP1 de ST Microelectronic [93] et le Zynq 7000 de Xilinx [141]. Chacun de ces SoCs est équipé d'un processeur double cœur de type ARM Cortex-A et peut accueillir un OS complexe (Linux dans notre cas). Le STM32MP1 a la particularité d'inclure un microcontrôleur de type ARM Cortex-M en plus du processeur applicatif, ou Application Processor (AP) en anglais, afin d'y déléguer les opérations de temps-réel, de communication externe ou d'entrées-sorties.

À partir de ces deux SoCs, nous décrivons trois scénarios attaquant-victime illustrés dans la figure 3 : une attaque SbSCA d'un cœur applicatif AP vers un autre sur le Zynq (figure 3.a). Puis des attaques du cœur applicatif AP vers le microcontrôleur MCU (figure 3.b) et du MCU vers le AP sur le STM32MP1 (figure 3.c). Ces attaques sont menées sur divers algorithmes de chiffrements tels que l'AES OpenSSL [106] et



a) AP-vs-AP Attack b) MCU-vs-AP Attack c) AP-vs-MCU Attack

Figure 3 Les trois scénarios d'attaque SbSCA utilisant des delay-lines menées dans le chapitre 4

diverses implémentations de Rivest–Shamir–Adleman (RSA). De plus, les DLBs et les DLLs sont également utilisés pour mettre en place des canaux de communication cachés (covert-channels) entre processus dans la section 4.4.1.

À travers les expérimentations menées à l'aide des DLLs et les DLBs, il apparait que la consommation électrique du processus cryptographique victime peut être collectée dans chacun des trois scénarios. Avec une fréquence d'échantillonnage des capteurs d'environ 16 MHz, il est possible de visualiser la consommation électrique des AES testés et l'exponentiation modulaire du déchiffrement RSA. De plus, nous démontrons pour chaque scénario, que des attaques CPA peuvent être menées et accomplies sur des implémentations cryptographiques non-protégées contre la SCA.

Ces attaques réalisées en utilisant uniquement les ressources matérielles contenues dans les circuits ciblés sont les premières à viser des systèmes aussi complexes (auparavant des microcontrôleurs avaient été visés [46], [104]). La présence d'un OS complique l'attaque car celui-ci génère un bruit de consommation et de la désynchronisation temporelle. Cependant, elle ne l'empêche pas. Au final, les résultats obtenus sont les suivants:

- *Attaque AP-vs-AP* : la clé d'un AES OpenSSL est récupérée en 20 millions de courbes en utilisant les DLLs dans un SoC Zynq 7000. (12 heures)
- *Attaque MCU-vs-AP* : la clé d'un AES OpenSSL s'exécutant sur un OS Linux est récupérée en 40 millions de courbes en utilisant les DLBs du STM32MP1. (24 heures)
- *Attaque AP-vs-MCU* : la clé d'un AES OpenSSL s'exécutant sur le microcontrôleur est retrouvée en 10 millions de courbes en utilisant les DLBs du STM32MP1. (9 heures)

Ces travaux menés sur deux SoCs modernes participent à démontrer l'ampleur potentielle de la menace des SbHA et ont été publiés dans *SideLine* [53]. Il semble désormais acquis qu'une grande majorité des SoCs peuvent être soumis à ce type d'attaque. Dans la conclusion du chapitre 4 nous décrivons les méthodes qui peuvent permettre à un pirate

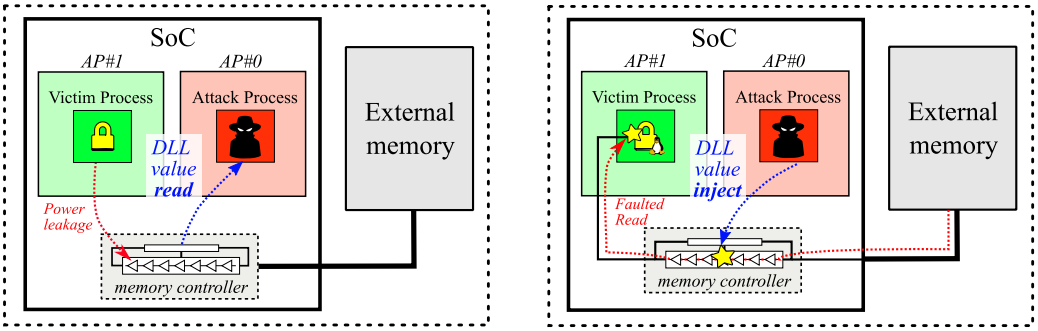
de détecter les ressources matérielles des SoCs nécessaires aux attaques SbSCA. Nous identifions trois méthodes de détection principales :

- *Les commandes OS exposées à l'utilisateur* : La plupart des systèmes complexes comportant un OS exposent des commandes utilisateurs dont certaines sont dédiées au contrôle de ressources matérielles (comme les paramètres Dynamic Voltage and Frequency Scaling (DVFS) dans [128]) ou à l'accès à des mesures de température ou de tension provenant de capteurs intégrés (comme la commande non privilégiée `powercap` dans [85]). Ces commandes offrent un accès direct aux ressources matérielles et ne nécessitent pas de réelle connaissance interne du système. Une attaque exploitant ces interfaces est donc relativement simple à mettre en place. Cependant, ces interfaces sont souvent loin d'être optimales en terme de flexibilité ou de vitesse d'accès (l'interface `powercap` ne peut être accédée qu'à une fréquence de 20 KHz dans [85]). Une autre méthode plus efficace consiste à accéder directement les modules impliqués dans l'attaque en utilisant leurs adresses physiques. Elle est décrite ci-dessous.

- *La documentation matérielle* : Dans le chapitre 4, les DLBs et les DLLs ont pu être identifiés grâce à l'existence d'une documentation détaillée décrivant chaque module matériel des SoC étudiés. Cette méthode qui se base sur la présence d'une documentation précise permet d'accéder directement aux registres permettant de réaliser l'attaque. Même si le travail de documentation est plus conséquent, ceci permet d'obtenir de bien meilleures performance pour les vecteurs attaques (l'accès aux delay-lines dans le chapitre 4 se fait à une fréquence de 16 MHz).

- *L'ingénierie inverse de code* : Dans le cas où la partie matérielle du SoC est peu ou pas documentée, il est possible d'étudier les modules noyau de l'OS qui sont chargés de l'accès aux ressources matérielles du SoC afin de retrouver leurs adresses physiques. L'utilisation d'outils d'ingénierie inverse tels que Ghidra [101] ou Radare [108] sur des modules noyau peut permettre l'extraction de ces adresses et le montage d'attaques SbHA plus performantes.

Afin de détecter et corriger des failles potentielles dans les SoCs, il semble important d'étudier ces trois méthodes d'accès aux vecteurs d'attaque. On remarquera que la mise en place des SbHA demande, au-delà des compétences en sécurité matérielle, une connaissance accrue des OS, des logiciels et du reverse-engineering. Ainsi, il semble que la conception de contre-mesures face à ces attaques nécessitera aussi l'expertise combinée d'experts en sécurité logicielle et matérielle.



a) SideLine Attack b) FaultLine Attack

Figure 4 De *SideLine* à *FaultLine*

Attaques par injection de faute dans les transferts mémoire des SoC complexes

La figure 4.a illustre le fonctionnement de l'attaque *SideLine* menée dans le chapitre 4 où une application malicieuse accède les DLLs pour collecter la fuite de consommation induite par un processus victime s'exécutant en parallèle. Dans le chapitre 4, le but principal des DLLs et des DLBs qui est d'assurer une bonne communication entre le SoC et des mémoires externes n'est pas pris en compte. Les attaques réalisées se concentrent uniquement sur la relation entre l'état de la delay-line et la consommation électrique du circuit. Néanmoins, la modification du calibrage des DLLs et des DLBs pourrait avoir des effets désastreux sur les opérations de chargement et de stockage mémoire opérées par le CPU.

Dans le chapitre 5, nous décrivons une autre attaque utilisant les delay-lines comme vecteur principal. La différence avec le chapitre 4 est que la delay-line est maintenant utilisée comme un moyen d'attaque par fautes SbFIA. Cette variante modifie donc le vecteur d'attaque SbSCA en un moyen d'attaque SbFIA sur les transferts mémoires.

Comme illustré dans la figure 4.b, cette attaque utilise une application malveillante pour modifier la calibration de la DLL afin d'injecter des erreurs d'accès mémoire dans des applications s'exécutant en parallèle. Ce travail introduit une méthode de SbFIA inédite qui s'attaque uniquement aux transferts mémoire et qui pourrait être menée dans une grande partie des processeurs utilisant des mémoires externes.

Les travaux menés dans ce chapitre ont été évalués expérimentalement sur un SoC Xilinx Zynq 7000 déjà utilisé dans les chapitres précédents. Nous montrons tout d'abord que les DLL peuvent en effet être recalibrées pendant un cours instant et induire des erreurs d'accès mémoire dans un processus. Nous réalisons ensuite plusieurs attaques d'un processus vers un autre. La première est décrite dans la section 5.3.2. Elle consiste à fauter le chargement de données de la mémoire DRAM vers un processus victime. Pour certaines valeurs de calibration de la DLL des fautes exploitables apparaissent. Les valeurs de faute optimales sont décrites dans la section 5.3.1.1.

À la suite de travaux préliminaires menés sans OS (en "bare metal") des scénarios d'attaques sont montés d'une application vers une autre (sur un OS Linux). Ces travaux menés dans la section 5.3.3 comportent l'extraction de la clé secrète de plusieurs types

d'algorithmes de chiffrement. Une attaque par faute persistante, ou Persistent Fault Attack (PFA) en anglais, [145] est menée à bien pour récupérer la clé d'une application de chiffrement basée sur le TinyAES [78] ainsi qu'une attaque Bellcore [15] sur une version non-sécurisée du RSA OpenSSL.

Ces travaux ajoutent la dimension SbFIA à cette thèse et confirment une fois de plus les dangers liés à l'accès malicieux aux ressources matérielles partagées des SoCs. L'attaque *FaultLine* relatant ces expérimentations a été publié dans [51].

Dans la conclusion de ce chapitre nous traitons l'apparition possible des attaques SbHA à grande échelle. Nous rappelons que la mise en place d'attaques SbHA est souvent plus complexe que la mise en place d'attaques logicielles classiques puisque les premières dépendent du matériel implémenté sur la cible. Ces attaques privilégient donc les vecteurs matériels génériques telles que les mémoires DRAM pour Rowhammer ou les delay-lines pour *SideLine* et *FaultLine*. Une fois qu'un malware exploitant ces vulnérabilités aura été mis en place, il pourra être propagé de la même façon que les attaques logicielles sur des milliers de plateformes.

Conclusion et Perspectives

Le chapitre 6 conclut ce manuscrit. Il revient sur l'impact académique et industriel de la thèse et la question de recherche abordée dans ce manuscrit. Dans la liste ci-dessous nous rappelons les contributions de cette thèse :

- Le chapitre 3 contient 3 travaux étudiant la menace des attaques SbSCA menée sur des FPGAs. Ces expérimentations ont donné lieu à la publication de deux articles de conférence présentant de nouveaux capteurs digitaux pour les attaques SCA sur FPGA [50], les premières attaques SCA statistiques d'un FPGA vers un CPU [54] et la mise en place d'un outil open-source d'évaluation de ces attaques [52].

- Les expérimentations menées dans le chapitre 4 ont donné lieu à la soumission de deux brevets et à la publication de l'article *SideLine* présentant les premières attaques SbSCA menées sur des SoC complexes [53].

- Les expérimentations menées dans le chapitre 5 ont donné lieu à la publication de *FaultLine* présentant des attaques SbFIA menées sur les transferts mémoires de SoC complexes [53].

À travers l'étude de faisabilité des SbHA, l'identification de nouveaux vecteurs d'attaques et l'exploration de nouveaux scénarios de SbHA, nous avons cherché à répondre à la question principale derrière ce sujet thèse : Faut-il considérer les SbHA comme une menace sérieuse pour la sécurité des circuit intégrés ?

Les expérimentations menées tout au long de ce manuscrit ont permis d'évaluer les SbHA sur divers appareils (CPU, MCU, FPGA), nous avons découvert de nouveaux vecteurs d'attaque SbFIA et SbSCA à travers l'identification des delay-lines, et enfin,

nous avons utilisé ces vecteurs pour construire de nouveaux scénarios d'attaques SbHA. La somme des résultats démontre que les attaques SbFIA et SbSCA sont effectivement capables d'extraire des secrets d'un appareil mais également pertinentes dans une grande variété de systèmes connectés. Il apparait que tout accès matériel laissé ouvert à un logiciel malveillant peut conduire à une divulgation complète du contenu d'une cible. Il est donc urgent de capitaliser sur les enseignements tirés de cette thèse et d'adopter les contre-mesures proposées dans chaque chapitre d'expérimentation pour éliminer efficacement et durablement la menace des SbHA.

D'un point de vue industriel, nos découvertes ont été présentées au JIL Hardware-related Attacks Subgroup (JHAS) un conglomérat de laboratoires d'évaluation sécuritaire, de fabricants de circuits intégrés et d'autorités de certification nationales. Les acteurs du JHAS ont largement reconnu la menace SbHA et cette communication a contribué à l'adoption des SbHA dans les évaluations Critères Communs. Par conséquent, elles seront automatiquement incluses dans le nouveau catalogue d'attaques qui sera pris en compte pour les évaluations dans le cadre du futur schéma de certification sécuritaire européen.

Sur le plan académique, la nouveauté de l'expérimentation menée a conduit à la publication de plusieurs communications scientifiques. Un total de quatre articles a été publié et présenté dans des conférences internationales (CARDIS 2019, ReConFig 2019, COSADE 2021, HOST 2021). Ces travaux ont également été présentés dans divers séminaires à travers la France et l'Europe comme l'Asset meeting 2019 à Munich, le séminaire iMath 2020 à Toulon et le séminaire DGA INRIA 2021 à Rennes.

Finalement, la multiplication et la complexification des circuits intégrés associés à l'intégration d'entités de sécurité dans les SoC devrait conforter la multiplication des SbHA dans le futur. Pour contrer efficacement ces nouvelles attaques de plus grands efforts devront être mis en place par les fabricants de circuits intégrés et les développeurs d'OS afin de construire des systèmes résistants. Fort heureusement, les contre-mesures déjà implémentées dans les cartes à puce pourraient suffire à court terme à défendre les systèmes connectés contre de telles attaques. Pour ce faire, une véritable prise en compte de cette menace doit être opérée rapidement par l'ensemble des acteurs de la filière des circuits intégrés.

Chapter 1. Introduction

Abstract

This chapter lays out the objectives, scope, and structure of this thesis, and describes the methodology adopted to identify and build remote hardware attacks. After a brief historical overview of the computer security origins, it presents the new challenges that have arisen in recent years with the increasing connectivity and complexity of integrated circuits. It also describes the adopted work plan and the outline of this manuscript.

Chapter Contents

1 Introduction **1**

1.1 Thesis Context 2

1.2 Thesis Objectives 3

1.3 Roadmap: *From FPGA to CPU exploits* 3

1.4 Contributions 4

1.4.1 Software-based Power Analysis Attacks on FPGAs 4

1.4.2 Software-based Power Analysis Attacks on Complex SoCs 5

1.4.3 Software-based Fault Injection on SoC External Memory Transfers 5

1.5 Outline 6

1.1 Thesis Context

In 1958, the first Integrated Circuit (IC) was created by Jack Kilby from Texas Instruments. Barely eleven years later, digital computers made out of thousands of transistors brought the humans to the moon. Since then, the semi-conductor industry conquered almost every aspects of human interactions: communication, work, transport, etc. The 1965 Moore's law projection predicting a transistor's doubling every year in silicon chips has been fulfilled for 55 years in a row and today's customer grades processors contain tens of billions of transistors.

If the discovery of ICs was an instant revolution, their need for security became obvious thirty years later with the apparition of customers services based on this technology such as internet, personal computers or mobile phones. For a long period however, the IC itself was considered as inviolable and most of the security research focused on cryptanalysis, software and network vulnerabilities. The works conducted by Van Eck [36], Biham [13] and Kocher [77] during the eighties and nineties proved that assumption to be wrong. These researchers along with others, introduced various techniques such as Fault Injection Attack (FIA) and Side-Channel Analysis (SCA) to retrieve secrets from ICs by taking advantage of hardware vulnerabilities. These discoveries led to the creation of a new research field: hardware security.

Hardware attacks generally require the targeted IC (e.g smart-card, microcontroller or system-on-chip) to be directly interfaced with hardware attack tools (e.g laser, electromagnetic probe, or power glitch injector). For this reason they are often considered as local attacks. Because the equipment required for building such attacks is somewhat expensive, hardware attacks are often conducted by specialized academical or private laboratories and remain quite obscure for the general public. Yet, hardware attacks are powerful attacks able to extract secrets from the most secure devices. For this reason they are systematically employed to assess the security of critical products such as smart cards, passports or crypto-wallets.

Hardware attacks have often been opposed to software attacks that can be launched remotely, are scalable and cheap. Because software attacks do not require specialized equipment, they are also better known to the hacker community, and most systems have been compromised by flaws in their software implementation. However, in the recent years this frontier between hardware and software security worlds have been rapidly eroding. The IC complexification and their widespread connection to the internet has led to the apparition of software-hardware combined attacks that exploit hardware vulnerabilities through software code. This became possible as modern high-end ICs enable hardware reconfiguration and monitoring from software. This software-hardware cooperation is now maliciously exploited to conduct remote hardware attacks.

In this PhD thesis, we sought to assess the threat that remote hardware attacks could pose and the additional challenges that integrated security solutions will face in the near future. Our main goal was to evaluate products that could potentially be targeted by this

new type of attack such as Internet of Things (IoT) devices, smartphones products and cloud datacenters. We assessed the feasibility of various remote hardware attack threat models and discovered new scenarios that could be exploited remotely by attackers and launched on a large number of devices simultaneously.

1.2 Thesis Objectives

The main contribution of this PhD thesis is to demonstrate that traditional hardware attacks do not necessarily require direct physical access to a device and that they can be carried out on remote targets. We achieve this through the use of a Software-based Hardware Attack (SbHA), namely a malware that controls embedded hardware components to implement SCA or FIA attacks. The main objectives of this thesis are listed and described below:

1) Assessing SbHA Feasibility: At the end of 2018, the start date of this thesis, SbHA was in its infancy. Rowhammer attacks was the only topic that had been extensively studied, ClkSCREW had been published one year before and Field-Programmable Gate Array (FPGA)-based power SCA attacks were just emerging. At first, the hardware security community struggled in apprehending and evaluating the extents of the remote hardware attack threat. Several questions arose: is it feasible under realistic attack conditions? Is it more or less powerful than local attacks? Can it be mitigated using traditional countermeasures? Which systems are potentially targeted? This thesis aims at answering these questions and at assessing the remote hardware attack threat in general.

2) Identifying Generic SbHA Vectors: In 2018, a large majority of today's SbHA vectors hadn't been discovered. While Software-based Fault Injection Attack (SbFIA) was already carried out on various processors, Software-based Side-Channel Analysis (SbSCA) attacks were still bounded to FPGA devices and it looked like a lot of discovery could be made on this field. The research conducted in this thesis led to the discovery of novel SbHA vectors that we used both for SbSCA (*SideLine* in chapter 4) and SbFIA (*FaultLine* in chapter 5).

3) Exploring New SbHA Scenarios: Additionally to introducing new SbHA vectors, we used them to build novel attack scenarios. Among the wide variety of potential SbHA scenarios, we evaluated various possibilities such as FPGA-to-FPGA attacks, FPGA-to-Computer Processing Unit (CPU) attacks, CPU-vs-CPU attacks or CPU-vs-Microcontroller Unit (MCU) attacks. By covering this large spectrum we aimed at further demonstrating the extents of the remote hardware attack threat and at proving that any multi-user or multi-security domain system could be a potential target.

1.3 Roadmap: *From FPGA to CPU exploits*

As a starting point for this thesis, it was chosen to orient the research toward SbSCA as it was less represented and studied in the literature. We adopted an iterative work plan that

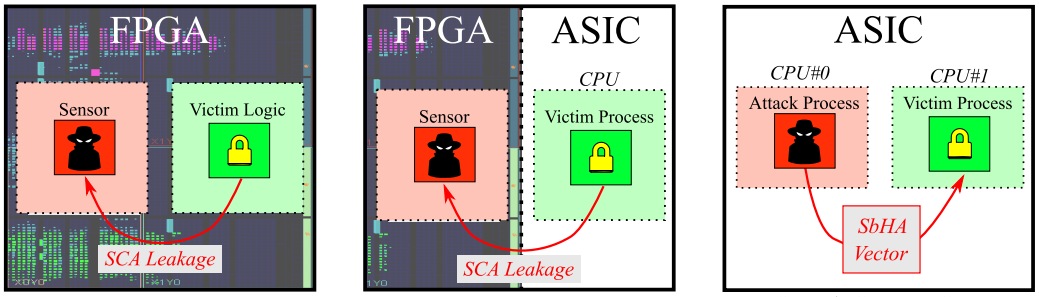


Figure 1.1 Thesis Roadmap

started from the reproduction and the enhancement of existing SbHAs to the creation of new attacks and mitigation methods.

Figure 1.1 illustrates the main steps of the thesis roadmap as it was described in 2018. We based our research works on the FPGA-based SCA attacks [122, 121] that were published a few months before by Schellenberg et al. This first step consisted in reproducing the few existing FPGA-to-FPGA attack works and bringing innovations to this topic. Then, we wanted the thesis to evolve from FPGA to Application Specific Integrated Circuit (ASIC) exploits in order to create more generic and reproducible attacks. Therefore, the second step naturally consisted in attacking heterogeneous platforms embedding both an FPGA and an ASIC within the same silicon die (FPGA-to-CPU attack). Finally, the last step had to occur on devices that didn't embed any reconfigurable logic. Therefore, it implied the discovery of novel hardware components (SbHA vectors in Figure 1.1) suitable for SbSCA or SbFIA in ASIC devices. These three steps led to various contributions that we describe in the following section.

1.4 Contributions

The steps identified in the 2018 roadmap led to three major contributions that will be described as chapters of this manuscript. The first contribution is linked to the implementation of FPGA-to-FPGA and FPGA-to-CPU attacks (step 1 and 2). This work includes the design of FPGA-based-delay sensors and their use for conducting SbSCA. The second and third contributions are applied to ASICs and disclose novel SbSCA and SbFIA methods that leverage delay-lines as SbHA vectors. We further detail these three contributions in the following sections.

1.4.1 Software-based Power Analysis Attacks on FPGAs

The first step of this thesis was to reproduce the few existing FPGA-based power SCA attacks. Through our experiments conducted using several algorithms and sensors, we observed that some sensor designs could be highly improved. This led us to the creation of new FPGA-based delay sensors designed to be very efficient for on-chip power SCA.

This contribution was published in the ReConFig 2019 conference proceedings [50].

The second step of this thesis was to move from FPGA-to-FPGA attacks to FPGA-to-CPU attacks. To that end, we evaluated heterogeneous ICs (embedding both a CPU and a FPGA) and captured the CPU leakage using FPGA-based sensors. This work was the first to implement FPGA-to-CPU attacks on symmetric cryptography. Many challenges were addressed such as the sensor limited sampling rate and resolution problem, the desynchronisation brought by the CPU and the noisy System-on-Chip (SoC) environment. In the end, we demonstrated that despite the challenges, such complex SCA attacks were feasible on these platforms and represented a major threat for the privacy of their users. Moreover, the SbHA knowledge gained during these experiments was decisive to later mount ASIC attacks. This contribution was published in the CARDIS 2019 conference proceedings [54] and its implementation was later released in an open-source framework paper [52]. We present this work in chapter 3.

1.4.2 Software-based Power Analysis Attacks on Complex SoCs

To fulfill our work plan and implement SbHA on complex ASICs, we had to find an alternative to FPGA-based sensors that could be directly accessible from the CPU. During our researches, we found out that delay-lines (which were already used in FPGAs to build voltage sensors) were also widely implemented in recent SoCs to handle external memory transfers. We studied these components and discovered that we could exploit their relationship with the chip's voltage fluctuations to conduct SbSCA. We called this SCA vector *SideLine* and used it to build the first SbSCA attacks on complex SoCs, that is, devices that can embed rich operating systems like Linux or Android. Several scenarios were implemented such as CPU-vs-CPU, CPU-vs-MCU and MCU-vs-CPU attacks. This contribution was published in the COSADE 2021 conference proceedings [53] and led to the application of two patents. We describe this work in chapter 4.

1.4.3 Software-based Fault Injection on SoC External Memory Transfers

Our discovery of delay-lines for building SbSCA led us to another variant. We observed that because delay-line settings can be accessed and modified during run-time and that they may be maliciously used to corrupt memory transfers. We then turned the passive SbSCA vector into an active SbFIA medium and built the *FaultLine* attack. By leveraging the granted access to delay-lines in various SoCs, we were able to induce glitches in external memory accesses and more significantly to corrupt data transfers, signatures and even to retrieve keys from cryptographic applications. This contribution was published in the HOST 2021 conference proceedings [51]. We describe *FaultLine* in chapter 5.

1.5 Outline

The remainder of this thesis is organized as follows:

- Chapter 2 describes the background on hardware attacks and provides a remote hardware attack classification.
- Chapter 3 analyses remote side-channel vulnerabilities in FPGAs and heterogeneous SoCs.
- Chapter 4 demonstrates that the logical isolation between physical cores in modern SoC systems can be defeated through SbSCA. It introduces *SideLine* an attack that uses delay-lines to collect on-chip power side-channel leakage.
- Chapter 5 introduces the concept of SbFIA on SoC external memory transfers. It describes *FaultLine* an attack that uses delay-lines to corrupt data read or write from/to external memories.
- Chapter 6 concludes the thesis and provides perspectives on future research directions.

Chapter 2. Background

Abstract

The title of this thesis "*Remote Hardware Attacks on Connected Devices*" introduces three main terms: *remote*, *hardware attacks* and *connected devices* that this chapter endeavors to define. After describing the major concepts behind hardware attacks and presenting connected devices and their vulnerabilities, we outline their recent status of potential victims of remote attacks. We establish a classification of remote hardware attack types and provide an in-depth state-of-the-art of the recent exploits and countermeasures published in the literature.

Chapter Contents

2 Background **7**
2.1 Introduction to Hardware Attacks 8
2.1.1 Origins 8
2.1.2 Attack Classification 8
2.1.3 Non-Invasive Attack Setup 10
2.1.4 Non-Invasive Side-Channel Analysis Attacks 11
2.1.5 Non-Invasive Fault Injection Attacks 14
2.2 The Advent of Connected Devices 16
2.2.1 Overview 16
2.2.2 Applications and Threats 16
2.2.3 Hardware Attacks: No Future? 18
2.3 Remote Hardware Attacks 18
2.3.1 The Origins of Remote Hardware Attacks 18
2.3.2 Remote Hardware Attack Families 24
2.3.3 Software-based Hardware Attack Taxonomy 29
2.3.4 Software-based Fault Injection Attacks 30
2.3.5 Software-based Side-Channel Analysis Attacks 37
2.3.6 Software-based Hardware Attack Privileges 42
2.4 Conclusion 45

2.1 Introduction to Hardware Attacks

Hardware attacks encompass all the attacks that use physical means to retrieve secrets from a device. They exploit physical properties intrinsic to Integrated Circuit (IC) components such as transistor's power consumption or ElectroMagnetic (EM) emanations to modify a device's behavior or eavesdrop its activity.

2.1.1 Origins

By the end of the twentieth century, the invention of smart cards offered a convenient method to integrate security and secrets within ICs. This plastic card's major objective was to provide to companies a way to identify customers and to decide whether they could access to a service or not. They were massively used in public telephone boxes, public transports and pay-TVs applications and are still prevalent today for banking (payment cards) and telecommunications (Subscriber Identity Module (SIM) cards).

Because smart cards were used to enable paid services, they were rapidly targeted by criminal organizations. For instance, the first individual smart cards used to decrypt TV channels were easily defeated because they were badly or poorly protected [82, 29]. Through simple software attacks using the smart card communication interface, the attackers were able to reverse-engineer the system access mechanism, conduct firmware updates and duplicate the exploit on blank cards. These counterfeits cards were then sold to clients for a much lesser price than the actual subscription to the TV channels [1]. A business was born.

A race between security updates and security exploits followed these events with a clear advantage for the attackers at the beginning. After a while, it became clear that "*Every security microcontroller and ASIC will be reverse engineered within weeks if pirates see a chance to make a million dollars profit from doing it*" [82]. Pirates tampering techniques evolved with the level of security implemented in smart cards. Once software attacks became more difficult to achieve because protocols had been secured, they started to disassemble the cards to extract secrets stored in the hardware. This was the starting point for hardware attacks.

2.1.2 Attack Classification

Hardware attack techniques have come a long way since the first experiments conducted in the nineties by Biham [13] on Fault Injection Attack (FIA) and Kocher [77] on Side-Channel Analysis (SCA). Since then, various attacks tools and methods have been designed to extract secrets from an IC. Nowadays, the hardware attack field is usually split into three main attack groups. From the least to the most intrusive: non-invasive attacks, semi-invasive attacks and invasive attacks [125]. Figure 2.1 depicts some differences

[1] https://www.wired.com/2008/05/tarnovsky/

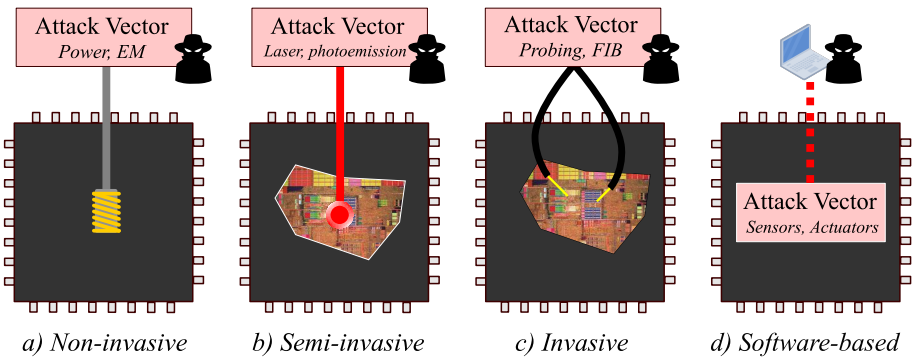


Figure 2.1 Proposed hardware attack classification

between each attack category.

- A **non-invasive attack** (fig 2.1.a) uses external equipment such as power and EM probes to collect the physical leakage induced by the transistor's activity of an IC (SCA) or to inject errors leading to faulty IC computational results (FIA). A non-invasive attack aims at retrieving the secrets securely stored within an IC without modifying or destroying it.

- A **semi-invasive attack** (fig 2.1.b) requires a partial depackaging of the targeted IC to expose the silicon die. This method is typically used in laser fault injection attacks [126] and in photo-emission microscopy [125] to establish a direct visual contact between the spot and the die. It is called "semi" invasive attack as it neither requires any physical contact with the internal wires nor chip destruction.

- An **invasive attack** (fig 2.1.c) requires the depackaging of the targeted device but also a direct physical access to the internal silicon chip. It uses micro-probes to tap the internal wires and collect information about the chip operations [63] (reverse engineering) and focused ion beams to cut the metal interconnections and build new ones [79] (modification attacks).

For several reasons, the attacks conducted in this thesis cannot be classified in any of the existing categories. Indeed, they differ from usual hardware attacks since they don't involve any equipment beyond the target itself. In that way, they enable hardware attacks on remote devices. However, despite the differences, our attacks exploit the same vulnerabilities linked to the physical properties of semiconductors. For this reason, we believe that the attacks presented in this thesis should take part in this classification but as a fourth category.

The name *software-based hardware attack* was proposed in [85] to identify this new kind of remote hardware attack and it seems to have been widely accepted by the community.

- A **software-based hardware attack** (fig 2.1.d) is triggered by a malicious program running within the target chip that aims at accessing software-exposed hardware mechanisms (sensors, actuators) in order to conduct hardware attacks such as FIA or SCA. Because a Software-based Hardware Attack (SbHA) exploits the target's physical properties, it may break the security barriers that normally prevent an application from accessing the information belonging to another application. By breaking this isolation,

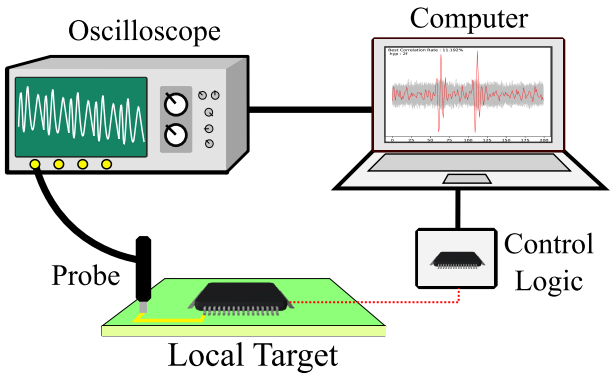


Figure 2.2 Local Non-Invasive Attack Setup

an adversary may steal any data stored within a device regardless from its privilege level.

Even if they don't share the same tools, non-invasive and SbHA leverage similar hardware vulnerabilities. In particular, they both use techniques known as FIA and SCA attacks. The following sections describe how non-invasive hardware attacks use these methods to extract secrets from electronic devices. It also describes how they can be translated into SbHA in some cases.

2.1.3 Non-Invasive Attack Setup

Figure 2.2 illustrates a typical non-invasive setup employed by laboratories to conduct local hardware attacks. This setup contains five major components which are mandatory for building an SCA or an FIA attack.

The first element is the **target**. It can be a smart card, a microcontroller or a System-on-Chip (SoC) and it is usually identified as the Device Under Test (DUT). The attackers place **probes** on the DUT package or power pads to conduct either SCA or FIA attacks. When used for SCA, the probes are designed to eavesdrop the IC physical emanations (power, EM, temperature). When used for FIA, they are specifically built for injecting glitches (EM, voltage) or pulses (laser) that may induce errors in the computations performed by the DUT. An **oscilloscope** is required for collecting, storing and visualizing the SCA traces. The acquired activity testifies of the computation in progress within the DUT and can be used in SCA for retrieving secrets or in FIA for synchronizing the attack with the target operations. The oscilloscope is the centerpiece of the attack bench. It has to precisely and quickly collect the probe state before sending the acquired traces to a remote **computer**/server for data storage and post-attack analysis. High-end oscilloscopes used for complex SCA attacks can achieve tens of GS/s sampling rates and an Analog-to-Digital Converter (ADC) resolution of 16 bits. This technology can cost tens of thousands of dollars. Finally, a **control logic** unit is required to enable interactions between the DUT and the computer. This entity (usually a microcontroller or an FPGA) translates the commands launched by the computer into a communication protocol understood by the DUT. It can transfer plaintexts, passwords and commands to interact with

the DUT and reset it on request.

Altogether, the DUT, the probe, the oscilloscope, the computer and the control logic constitute a minimal non-invasive attack setup. Several additional elements such as glitch injectors are sometimes added to the setup but are not required in every attack scenarios. In the following two sections we describe how this non-invasive setup is employed in laboratories to extract secrets from secure ICs.

2.1.4 Non-Invasive Side-Channel Analysis Attacks

As we have seen above, an IC is made out of transistors. These transistors have their output signal switching depending on the input voltage applied. They only take two states, high or low, 1 or 0, this is the so-called binary signal. Programs and applications running on top of the processors use millions of these transistors along with memory to handle complex computations, user interaction, graphic rendering, etc. Depending on the data computed, the transistor activity may highly fluctuate. The art of SCA attacks consists in eavesdropping this activity by collecting its side-effects on physical emanations such as power consumption, electromagnetic field, temperature, etc. Because they fluctuate with the calculations done by the processor, these side-channels may leak information on what is currently computed and lead to the extraction of sensitive data when a secret is computed.

2.1.4.1 Power Analysis Attacks

A power SCA makes use of the transistors switching activity leakage through power consumption variations to collect information about the processes running inside a target device. Thanks to the relationship between the transistor power consumption leakage and the data processed, a SCA attack can be performed to retrieve cryptographic keys from a target. Power side-channel setups rely on voltage probes and oscilloscopes to monitor the DUT side-channel leakage through a resistor attached to its power pads as depicted in Figure 2.2. By analyzing the collected traces, an attacker can visually speculate on the different operations performed by the device using Simple Power Analysis (SPA) [77]. Statistical side-channel methods such as Differential Power Analysis (DPA) [76] or Correlation Power Analysis (CPA) [18] allow an attacker to infer the secret keys of cryptographic processes by correlating guessed leakage hypotheses with a set of experimental traces.

Figure 2.3 represents the power leakage induced by a software Rivest–Shamir–Adleman (RSA) cryptosystem [116] implementation running within an IC. The captured RSA power trace delivers an information about the computations that were conducted by the processor. An attacker knowing the implementation details of the RSA will recognize this operation as a modular exponentiation. In this case of an unprotected RSA implementation, he would also be able to deduce if the key bits used

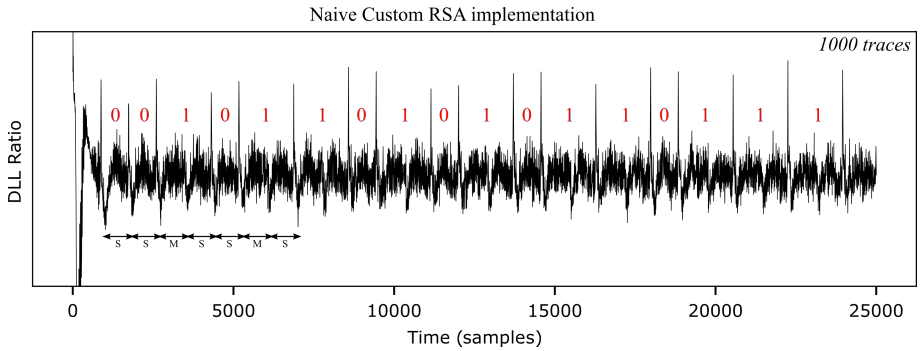


Figure 2.3 Simple Power Analysis on RSA

for the exponentiation were ones or zeros. Indeed, a key bit to '1' would induce more processor operations than a key bit to '0'. This can be directly observed by measuring the space between the black power spikes in Figure 2.3. With such a trace, the attacker would then retrieve the entire RSA private key in a short amount of time and thus compromise the device security. This attack is an example of SPA.

To tackle the SCA threat, security hackers work along with security architects to build efficient countermeasures. They are dedicated to complicate the key retrieval by making the signal less stable, less readable but also immune to statistical SCA methods. Countermeasures such as masking, jitter or shuffling [147, 149] are usually implemented in secure devices to prevent SPA, DPA and CPA but are often defeated by more complex SCA techniques such as higher-order statistical attacks [91].

Applications in Remote Hardware Attack (RHA): Power analysis attacks can be mounted remotely thanks to the presence of voltage sensors within modern processors. In 2013, Fujimoto et al. demonstrated on a custom Application Specific Integrated Circuit (ASIC) device that an on-chip measurement unit could collect the side-channel leakage of a cryptographic accelerator [38]. This leakage was then used to mount SPA and CPA attacks on Advanced Encryption Standard (AES) [30]. Since then, these attacks have been reproduced on various platforms and with different sensors. We address this topic in Section 2.3.5.

2.1.4.2 Electromagnetic Analysis Attacks

Electromagnetic emanations are another example of leakage that can be eavesdropped in ICs. When a device performs computations, the current running through the transistors generates a magnetic field and electromagnetic waves are emitted [40]. These can be captured using a coil connected to an oscilloscope and used to conduct equivalent attacks to those presented in the previous section. However, this time SPA becomes Simple ElectroMagnetic Analysis (SEMA) and CPA becomes Correlation ElectroMagnetic Analysis (CEMA) [113].

Electromagnetic analysis presents several advantages compared to power analysis. First of all, it doesn't require to find a power line, the probe coil is directly placed on top

of the DUT and the electromagnetic emanations can be collected through the package. Secondly, in power analysis, the presence of decoupling capacitors and voltage regulators in the DUT may filter leakage frequencies containing the footprint of the signal and alter the attack results. Because the EM probe is directly collecting transistor's EM emanations, the signal is less damaged by surrounding components and better reflects the actual leakage. Finally, this electromagnetic signal is often more convenient for side-channel traces resynchronization as it is less filtered and thus more sharp.

Despite the advantages of EM analysis, power analysis is still widely used as they are some applications where EM is hardly applicable. For instance, the emergence of 3D ICs with RAM and Computer Processing Unit (CPU) cores stacked on each other (package-on-package[2]) can thwart the EM analysis as the useful signal gets mixed up and hidden by the surrounding noise and as the probe is more distant from the silicon die. Moreover, in these complex ICs, distinct power lines exist for each hardware blocks (e.g., CPU, ADC, memory) and can then be separately analyzed through power analysis. This sometimes allows the attacker to probe the pad powering a specific block and may highly facilitate the analysis. To conclude, EM is often chosen for SCA as it is convenient and powerful but other methods such as power analysis continue to be used to replace EM when it can't be applied or to enrich the analysis with an additional leakage information.

Applications in RHA: EM radiations have been studied by the National Security Agency (NSA) since the cold war through the Telecommunications Electronic Material Protected from Emanating Spurious Transmissions (TEMPEST) specification as they could leak information at a quite distant range and therefore be captured remotely using an antenna [102]. In 2018, the subject has resurfaced in a paper demonstrating how mixed-signal ICs may leak EM side-channel information at a longer distance than usual [22]. We address this topic in Section 2.3.2.1.

2.1.4.3 Other Side-Channels

Other passive side-channels are also found in the literature such as optical, acoustic and temperature side-channels. These exotic side-channels are not usually employed in today's attack scenarios because they are less efficient than power and electromagnetic vectors. However, they have been demonstrated useful in some specific scenarios.

In [64] the thermal side-channel was practically evaluated and one of the conclusion was that "the temperature leakage is linearly correlated with the power leakage model but is limited by the physical properties of thermal conductivity and capacitance". Indeed, the temperature side-channel has a very low bandwidth which thus thwarts practical attacks. However, this paper also proved that thermal side-channel is sufficient to evaluate the power usage of a device. Several years later, thermal side-channel have found various applications. For instance, in [69], the researchers proposed to measure the temperature in multi-user tenants datacenters in order to detect opportunities for conducting Denial-

[2] https://en.wikipedia.org/wiki/Package_on_a_package

of-Service (DoS) attacks.

For their part, acoustic side-channels were used to exploit sounds emitted by electronic devices and related mechanical objects. Multiple acoustic works were published from the recovery of a text file using the sound made by a printer [10], to the extraction of RSA private keys through the sound generated by a computer during an encryption [42]. In practice, acoustic side-channels are more efficient for spying objects or peoples than for extracting cryptographic keys. Indeed, microphone bandwidth limitations to hundreds of KHz are usually too limiting to eavesdrop cryptographic activity on GHz frequency computers.

Despite their limitations, these other side-channels need to be taken into account by the security community as they will inevitably be feasible on certain systems. In this section we did not mention timing SCA, but it will be treated in Section 2.3.2.2.

2.1.5 Non-Invasive Fault Injection Attacks

In the semiconductor safety research field, it has been known for a long period that cosmic rays particles may induce soft errors in ICs [105]. These could for instance modify a value stored in memory and induce errors in critical computations. This ionization caused by cosmic rays could make a system erroneous, dangerous or even inoperative. For this reason, it was first considered as a significant safety issue for spatial and automotive applications.

In the nineties, researchers started to evaluate the semiconductor's physical interactions for security purposes. Smart cards were the first targets and they were rapidly defeated using fault injection techniques [13, 6]. In this section, we will discuss three types of non-invasive fault injection methods used to reveal IC secrets: clock, power and electromagnetic glitch fault injection.

2.1.5.1 Clock and Power Glitch Attacks

Clock and power glitch attacks have been used since the mid-nineties to induce errors within IC's computations. The idea is to inject a clock glitch (e.g., shorter clock pulse than normal) or a power glitch (e.g., transient in supply voltage) during a program's execution. If the glitch parameters are correctly set, this will result in inducing an error inside a program such as the execution of wrong instructions or the modification of a variable.

When applied to an unprotected device, this injection method may bypass security features such as Personal Identification Number (PIN) code verification or even lead to secret key extraction in cryptographic systems. Like SCA, FIA may use statistical analysis to extract keys from a device given a known algorithm and an injected fault. Famous techniques are known as Differential Fault Analysis (DFA) [13] or safe-error [127] attacks. They allow an attacker to retrieve the key of cryptographic algorithms through

fault analysis (e.g., Piret attack [110]). Other DFA methods such as Bellcore attack [15] can be used against public key algorithms (e.g., RSA).

Regarding the origin of the fault, it is widely accepted that it essentially comes from timing constraint violations in the internal circuit's logic paths [153]. In the case of a power glitch, a sudden decrease of the power supply (voltage drop) reduces the digital signal propagation speed through combinational gates. Because the system is synchronous, the signal must be ready before the clock edge (setup time). The lag induced by the voltage drop may result in an error if the logic signal does not have the time to reach the end of the combinational logic path before the flip-flop sampling. If the signal arrives too late, an unknown value will be fetched instead and will potentially induce an error. The clock glitch scenario is quite close but this time the data sampling is triggered too early in a way that the signal does not have the time to propagate through the logic gate before the data is sampled [2, 39].

Several countermeasures have been designed to thwart these injection mediums. For instance external clock sources such as quartz were rapidly abandoned for secure ICs to prevent the attacker from replacing it with clock glitch injectors. Clock signals are now generated internally using Voltage-Controlled Oscillators (VCOs) and inaccessible from the external pads. Moreover, various designs of internal glitch detection circuits have been proposed to detect power supply and clock anomalies [142, 143] and are implemented in smart cards to reset or even kill the card if multiple faults are detected.

Applications in RHA: Power and clock glitch injection attacks can be mounted remotely thanks to the presence of programmable voltage and frequency regulators within modern processors. In 2017, Tang et al. demonstrated that remote power and clock glitch attacks could be conducted on smartphones by exploiting software-exposed hardware regulators [128]. Since then, these attacks have been reproduced on various platforms and with different glitch vectors. We address this topic in Section 2.3.4.

2.1.5.2 Electromagnetic Glitch Attacks

EM fault injection was introduced in 2003 [119], but started to be intensively used ten years later thanks to significant improvements achieved in probe and injector designs [33, 98, 32]. Using a coil placed on top of a DUT package, one can inject an EM glitch to disturb the target's internal activity. The EM glitch injection can generate errors in running processes and lead to faults suitable for analysis and secret extraction. Again, the EM advantage over clock and power glitch injection is that it doesn't require access to the power or clock DUT pads. Moreover, the EM glitch is not directly affected by voltage regulators and decoupling capacitors that could partly absorb a power glitch. While power glitch would often require capacitors removal, the EM glitch can be conducted without any modification of the Printed Circuit Board (PCB). For these reasons, EM injection is the preferred method to inject glitches even though its configuration is a bit more expensive.

EM injection suffers the same limitations as EM side-channel. For instance, 3D pack-

ages or stacked RAM may protect the inner silicon against the EM glitch as it would need to get through all the layers to inject a fault. This results in the need for more powerful injectors on these targets and makes it difficult to understand the origin of the fault. Because complex and stacked semiconductor architectures will become more prevalent, expertise in both power and EM injection will remain mandatory in the future.

2.2 The Advent of Connected Devices

Following the description of the concept behind physical security and non-invasive attacks, the second part of this chapter focuses on the second major term of this thesis topic: “connected devices”. Here, we aim at better specifying the type of targets that this thesis would consider.

2.2.1 Overview

The emergence of the Internet at the end of the nineties has enabled a low-cost universal connection around the world. Personal computers were the first devices equipped with Ethernet ports, WiFi connection and also the first to be subjected to cyberattacks. Despite its benefits, the use of Internet introduced millions of malware inside computers. The reason was the poor security knowledge of the end-users and the lack of integrated security such as firewalls to tackle the attacks. Software security architects have been working together for thirty years to strengthen computer security but as for hardware attacks, it seems that nothing can prevent a motivated attacker from obtaining what he wants.

Recently, with the IC miniaturization and the improvements done in Internet bandwidth, connected devices have been invading almost all the human activities and services. While providing remarkable advances for industrial or medical systems, this breakthrough still suffers from its lack of security. In the next section, we introduce a sample of connected device applications that are currently targeted by cyberattacks and in the RHA scope.

2.2.2 Applications and Threats

Recently, industries, companies and public services have been widely adopting connected devices in order to improve their flexibility and efficiency. For instance, the fourth industrial revolution “Industry 4.0” promotes the digitization of manufacturing. To improve the productivity, machines should be constantly connected and monitored to create resilient and adaptable systems. In industries, one could improve the overall power efficiency by remotely controlling each engine and device. In hospitals one could better schedule the medical operations by anticipating the specialized machine usage.

This massive adoption of connected devices in countless sectors opened the door to a new business for software hackers. By designing malwares that can propagate throughout

internal networks, the adversaries are able to shut down entire websites or services. To access internal networks, hackers essentially rely on phishing email campaigns and software vulnerabilities [68]. Once inside, they can delete or encrypt the entire company data and ask for a ransom to get it back. According to the Cybercrime magazine, a ransomware is expected to attack a company every 11 seconds in 2021[3]. Their cost for businesses and states should reach 20 billion dollars in 2021. Considering the fact that a large part of today's systems are connected, this may result in interrupting company services for hours, days or months. In case of a hospital attack, the implications are serious.

Despite these threats, connected devices are more and more widespread. The Internet of Things (IoT) has been enabling Internet connections on smaller and smaller systems. Consumers applications have been soaring. The example of smart homes is striking with connected ventilation, lighting control, leak detection, home robots, smart kitchens, etc. All these objects were often left totally unprotected against cyberattacks and their critical applications raise some questions about our indoor security. A striking example was how an attack [118] could take control on smart light bulbs and trigger a chain reaction on thousands of devices located nearby. Here the hackers demonstrated the threat on a harmless bulb device, but the fact that it could be reproduced on leak detection systems or connected healthcare systems is alarming.

Simultaneously, society is also facing the dematerialization of various devices replaced with services. Cloud datacenters are emerging with the acceleration of the global Internet network and provide services to consumers and companies that want to store data or perform complex computations without having to deal with internal servers. Cloud services offer state-of-the-art equipment and maintenance that are really interesting for companies which cannot afford servers and technicians. The datacenters consist in big computers shared between multiple users: companies, people, governments. The most famous ones belong to huge companies such as Amazon[4], Google[5] or Microsoft[6]. As more and more end-user services are offered and private data stored, the security question becomes inevitable. Is the user data safe? Could we be spied by cloud providers? Could we be spied by other users?

During this thesis, we evaluated several scenarios of hardware attacks on connected devices. From IoT device attacks to cloud scenario attacks we imagined how attackers could try to steal data in these connected environments. The connectivity revolution is far from being free from security mistakes, that's why we strongly need to evaluate these systems against various and new types of attacks.

[3] https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021/
[4] https://aws.amazon.com/
[5] https://cloud.google.com/
[6] https://azure.microsoft.com/

2.2.3 Hardware Attacks: No Future?

Hardware attacks are often considered as local attacks because they require direct physical access to the target. Indeed, as described in Section 2.1.3, hardware attacks require physical material such as oscilloscopes, lasers or glitch injectors and are thus local by design. This characteristic limits the variety of devices that can be evaluated. Smart cards, microcontrollers and SoCs are the usual targets because they are small, they can be stolen and they are easy to place on a test bench. However, less accessible systems such as servers, datacenters or even buried and hidden devices cannot be evaluated.

With the advent of IoT, cloud services and decentralized computing we could question the future of hardware attacks. Indeed, if in a hypothetical future, all the cryptographic calculations for all the devices were done remotely in closed data centers, the field of possibilities for hardware hacking would eventually shrink.

This idea is biased as we consider these systems immune against hardware attacks. As we seen in the previous section, the security of connected devices is an issue of general order in all the human activities and services. These security concerns are not only related to the software embedded but also to the hardware that could be maliciously used to conduct attacks. This security by distance mirage could rapidly decay with the emergence of RHAs.

Since 2014, RHAs have been invading the academic conferences and journals. Novel attack vectors are disclosed every months and threaten the security of various systems. This new threat is a revolution in the hardware security world. It involves new skills, new methods, new analysis tools and also new countermeasures. The following section introduce their origins and the main families of RHAs.

2.3 Remote Hardware Attacks

Various types of attacks may be classified as RHAs. However they must match at least two conditions. First of all, they have to interact with the device hardware. Secondly, they need to be more distant than local techniques and do not involve stealing the target. An attack that is usually conducted at several millimeters from a device will be considered as remote if it is conducted at several meters. The difference doesn't seem that significant but can be a game-changer in terms of attack scenarios. By finding a method that works at a distance of 10 meters, the attacker would no longer need to steal the target to extract its secrets.

2.3.1 The Origins of Remote Hardware Attacks

The idea of launching hardware attacks remotely has been around for decades. During the cold war, the NSA already warned of the potential dangers of eavesdropping attacks in the TEMPEST specification [102]. The concerns here were that compromising EM

emanations could be collected remotely by an attacker using an antenna. The first public work exploiting TEMPEST vulnerabilities was published in 1985 by Van Eck [36] and described a method to extract the image displayed by a computer monitor from hundred meters away and using low-cost equipment. More recently air-gapped systems have been heavily targeted by RHAs exploiting various types of emanations such as optical, acoustic or electromagnetic ones [60, 62, 61].

In parallel, the complexification, the increasing connectivity and the integration of security features within modern ICs brought new hardware vulnerabilities directly exploitable from software. SbHA and microarchitectural hardware attacks use software to exploit hardware vulnerabilities. Because they only rely on software code, they can simultaneously infect many devices that are built on the same hardware architecture. These attacks exploit three phenomena that we will describe in the following sections: the increasing complexity of modern devices, the adoption of multi-user systems and the emergence of integrated security solutions.

2.3.1.1 The Increasing Complexity of Modern Devices

To better comprehend the mutations that occurred in the semiconductor industry lets compare two ICs. The first one is the Motorola 68000 [99] shipped in 1979 and the second one is the Apple M1 processor[7] shipped in the end of 2020. The Motorola 68000 was used in the first Apple Macintosh computer. The Apple M1 processor in used in the most recent Apple laptops.

Table 2.1 highlights the major evolutions that have been made 40 years after the Motorola's 68000 release. First of all, the transistor's process has shrank from 3500 to 5 nm leading here to a transistor count increase from 68 thousand to 16 billions. The clock speed also rose with a impressive 320 factor (10 to 3200 MHz). Recent systems use multi-cores architectures to support multi-tasking (here 8 cores for the Apple M1) and their data-width (bus size) is now reaching 64-bits. Considering these numbers, one could fit as much as 230,000 Motorola 68000 processors in the Apple M1 and it would still get lower performances.

	Motorola 68000 	Apple M1 
Process	3500 nm	5 nm
Transistor Count	68,000	16,000,000,000
Frequency	10 MHz	3.2 GHz
Data width	16/32 bit	64 bit
Number of cores	1	8

Table 2.1 The impact of 40 years of research in the semiconductor industry

The real difference between these two devices is that the M1 is way more than a

[7] https://en.wikipedia.org/wiki/Apple_M1

processor as it centralizes all the components required to build a computer. It indeed embeds a CPU but also a Graphic Processing Unit (GPU), an image processing unit, a digital signal processor, a neural processing unit, cache memories, integrated-security, a power management unit, etc. This type of device that implements various components on the same silicon die is usually called a SoC. It is way faster than older systems that used one chip per function as it reduces the travel times for communication between the components. The SoC architecture is becoming a standard in today's computer and microcontroller systems. Indeed, the performance gains obtained and cost saving realized are really attractive for semiconductor companies. This way of designing chips follows the "More-than-Moore" concept [138] that describes how modern devices do not only count on the "Moore's law" to increase their performances. Instead of relying only on the number of transistors, new ICs incorporate independent building blocks optimized for specific tasks on the same silicon die (power management, neural units, hardware acceleration, etc).

The emergence of these heterogeneous SoC systems also impacted the way software interacts with the underlying hardware. To meet the ever-growing need for performance in silicon devices, SoC providers have been increasingly relying on software-hardware cooperation. By controlling hardware resources such as power or clock management from the Operating System (OS), developers earn the possibility to build more flexible and power efficient applications. Despite the benefits, these hardware components are now exposed to software code and can potentially be misused as open-doors to new kinds of attacks.

2.3.1.2 The Adoption of Multi-User and Multi-Tasking Systems

The increasing complexity of ICs gradually enabled multi-tasking and multi-user platforms. That is, systems where different programs and users simultaneously share the same hardware resources. Today's processors can run hundred of applications simultaneously and handle several users with different OS privileges.

From a security point of view, running applications simultaneously is not an easy task. Strong memory isolation should be implemented to prevent one application from tampering with the others and a certain privilege level must be assigned to each application to make sure that an untrusted process doesn't access and modify critical IC/OS components. In multi-user systems such as cloud services, a Virtual Machine (VM) must be properly handled to prevent users from spying or disturbing each other. To that end, hypervisors are used to isolate guest VM from each other. The hypervisor is a piece of software in charge of controlling the underlying hardware that maps memory spaces, e.g., the Memory Management Unit (MMU). It has the highest privilege level as it can apply modifications on the underlying hardware to maintain the isolation between VMs. If the hypervisor is defeated, then the entire isolation is compromised and attacks can happen between users.

These security principles are widely implemented in personal computers, smart-

phones and IoT devices. The famous Linux[8], Android[9], Windows[10] and Apple[11] OS implement several security domains. For instance Linux splits the memory between a user space and a kernel space. The kernel space is where the core of the OS executes and provides its services. The kernel represents the highest privilege level, it enables direct access to the underlying hardware such as memory, I/Os and power management. The user space is where the user programs runs. It may have distinct permissions levels such as unprivileged user and super-user in Linux. User programs are handled by the MMU so they can't access each other. If a user program needs to access a physical resource (e.g., read I/O, memory or access a peripheral), it has to perform a system call to the kernel. Then, depending on the process permissions, the kernel will grant or discard the access to the specified resource.

Hundreds of different kernel versions are available for each OS distribution, some are designed for servers, some are designed for low-power devices, some are designed for security, etc. Depending on the OS implemented, an unprivileged user may gain access to critical physical components or may perform privilege escalation by exploiting kernel vulnerabilities. Because they are such complex environments, OS are continuously subject to attacks which need to be patched by the developers. Because the attackers still remain one step ahead in this security war, it becomes clear that rich OS are way too complex to be used as trusted platforms. This observation gradually led to the emergence of integrated security in modern ICs.

2.3.1.3 The Emergence of Integrated Security

Because rich OS attack surface is too large, they cannot be easily certified or trusted. For this reason, several solutions have been proposed by IC providers to discharge the hypervisors from several critical security operations. The idea here is to create secure areas that can thwart threats from the rest of the device (malicious OS, malwares, etc).

From SIM cards to Integrated SIM

Delegating secure operations to a specialized device is not new. It has been done for many years on mobile phones using SIM cards. A SIM card consists in a simple processor containing a specialized OS dedicated to securely connect a mobile phone to external cellular networks (3G, 4G, 5G) and Internet services. The SIM card stores phone numbers, contact information and SMS. Malicious data extraction from a SIM card can be challenging because these devices are designed by specialized companies to resist to state-of-the-art hardware and software attacks. Moreover, it is a standalone element separated from the mobile phone application processor. This physical separation highly limits the attack surface for the attackers and thus the potential threats.

[8] https://www.linux.org/
[9] https://www.android.com/
[10] https://www.microsoft.com/windows
[11] https://www.apple.com/macos/
[12] https://kigen.com/resources/blog/securing-iot-in-a-5g-world/

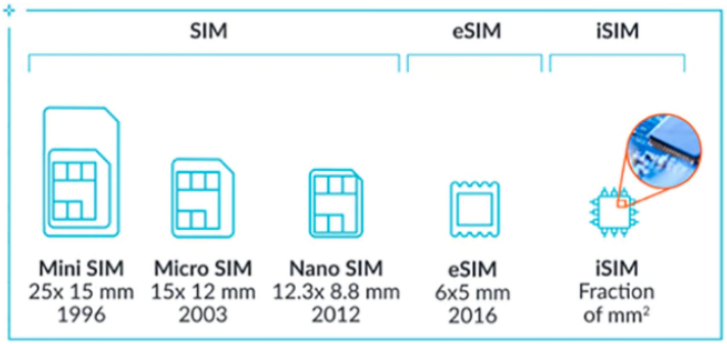


Figure 2.4 Evolution of SIM[12]

As we mentioned earlier, today's trend is to integrate various functionalities within a single device. This statement also affects the security elements and the SIM cards. Figure 2.4 illustrates the gradual SIM card miniaturization observed between 1996 with the Mini SIM and 2012 with the Nano SIM. Then, the emergence of the embedded SIM (eSIM) in 2016 which consists in a component directly integrated on the device's PCB. And, more recently, the apparition of the integrated SIM (iSIM) implemented as a block of the SoC system. In the future, the card format should gradually disappear and leave room for the eSIM and iSIM formats to dominate[13].

Integrating security components within untrusted devices has recently been subjected to controversies. Indeed, even if the secure components are still protected from malicious software accesses, they now share the same silicon die with other potentially malicious entities. Therefore, the risks of being subjected to RHAs is rising .

Securing the Insecure: Hardware-based Trusted Execution Environment

A lot of companies such as cloud providers, IoT device manufacturers or automotive constructors need more secure systems and for this reason integrated security as never been such a major selling point. A Trusted Execution Environment (TEE) is a secure zone in the processor. It runs in parallel with the main OS, in an isolated environment. It aims at defending sensitive programs and data against unprivileged and privileged software attacks from a potentially compromised native OS. In the recent years, processor manufacturers have been designing several hardware methods to strengthen TEE security. The Intel Software Guard Extensions (SGX) [67], the AMD Secure Encrypted Virtualization (SEV) [5] and the ARM TrustZone (TZ) [7] are some examples of the hardware solutions designed to bring security in computers and microcontrollers at a higher level. Despite some differences in their architectures, these three security mechanisms share their main functionality which is to isolate some programs from the insecure application OS. These solutions are heavily used in today's systems to protect VMs in the cloud and secure applications in smartphones. They act as adding an additional privilege level on top of the existing kernel and user spaces.

[13] https://iotbusinessnews.com/2021/03/20/69747-esim-technology-is-predicted-to-be-the-next-step-in-the-evolution-of-iot-devices/

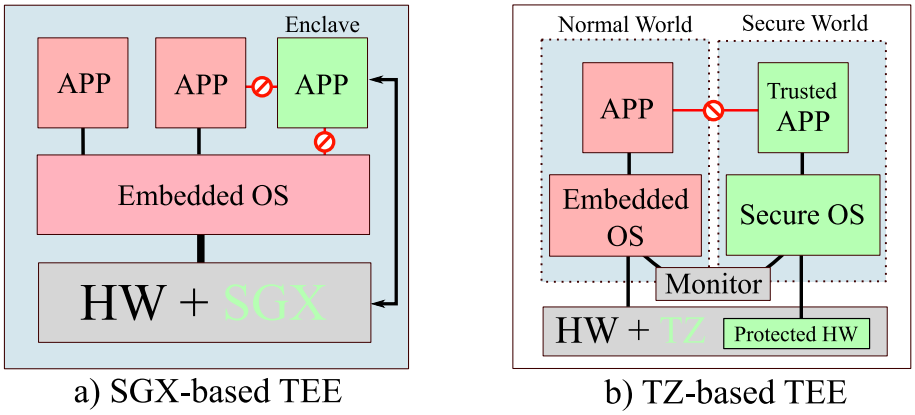


Figure 2.5 SGX and TZ-based Trusted Execution Environments

Figure 2.5 illustrates two types of TEEs. On the left, a typical Intel SGX-based TEE is represented. In red, the user applications and the OS are considered as untrusted. To enable the execution of a secure application the SGX hardware mechanism allows any user or kernel program to create enclaves (app in green). These are defined private regions of memory, whose content is protected and unable to be either read or saved by any process outside of the enclave itself. Moreover, the enclave content is encrypted and decrypted on the fly by the SGX hardware to prevent any attempt from dumping the enclave memory. SGX has been designed to strengthen security in various use cases. It is for instance used in secure cloud computing[14], secure web browsing[15], and Digital Right Management (DRM) [12].

On the right, a typical TZ-based TEE is represented where the processor and its hardware resources are shared between a secure world (green) and a normal world (red). When operating in the secure mode the CPU can access all of the device's hardware and memory (protected HW in green). When operating in normal mode, it can only access to a subset of peripherals and unprotected addresses in memory. The normal world contains the rich OS (Linux, Android) and the user applications. These entities cannot access the protected hardware area regardless of their privileges. The secure world contains trusted applications that provide critical services such as key storage, cryptographic operations or trusted authentication. They run on top of the secure OS whose behavior may differ depending on the TEE provider. At hardware level, a processor register bit defines if the system is running in normal or secure mode. Only the secure monitor component can modify this bit and change the mode of operation through a Secure Monitor Call (SMC). TZ-based TEEs are widely used on mobile, automotive and IoT platforms to secure critical operations such as fingerprint authentication, encryption services and mobile payments.

Despite the design differences, TZ and SGX share the same goal which is to protect secure operations from unprivileged and privileged malwares. These two security mech-

[14] https://www.ibm.com/cloud/blog/data-use-protection-ibm-cloud-using-intel-sgx
[15] https://software.intel.com/content/www/us/en/develop/articles/hardening-authentication-tokens-in-browsers-using-intel-software-guard-extensions.html

anisms even assume that the rich OS could be compromised. TZ and SGX-based TEEs efficiently tackle various software attack scenarios and their contribution in building more secure computer systems cannot be contested. However, by introducing new privilege layers on top of the kernel they also led to the apparition of novel attacks. SbHA which require access to internal hardware resources have found an additional use case in targeting these TEEs implementations. Even if hardware attacks are out of the scope of TZ and SGX, the fact that SbHA can be conducted remotely and on multiple targets make them as detrimental as software attacks and thus worrying for hardware TEE security.

2.3.2 Remote Hardware Attack Families

The increasing complexity of modern devices, the adoption of multi-user systems and the emergence of integrated security led to the apparition of RHAs. In this section we will briefly introduce three attack families that can be defined as RHAs: TEMPEST attacks, microarchitectural attacks and SbHAs.

2.3.2.1 TEMPEST Attacks

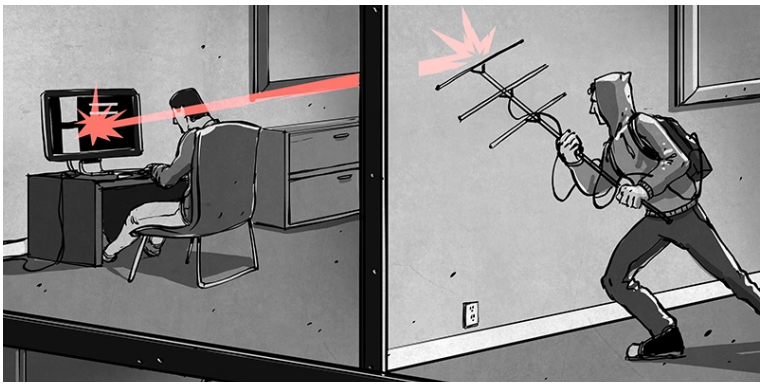


Figure 2.6 Artist's representation [16]of the TEMPEST attack threat

In Section 2.3.1, we mentioned the existence of the NSA TEMPEST specification that considered device emanations as a potential source of information leakage. The work on computer monitors started by Wim van Eck in 1985 was the first academic paper [36] to publicly evoke this danger and it paved the way to a new family of attack dedicated to the data extraction from remote devices. Since then, several works have been published and demonstrated that the compromising emanation sources are not only electromagnetic, they can also be optical, thermal and acoustic. From these emanations two families of attacks arose: remote covert channels and remote SCAs.

Several works leveraged remote covert-channels as a method to extract data from air-gapped computers. That is, devices that are kept isolated and disconnected from any external network to avoid information leak. These air-gapped computers are often used in

[16] https://hackaday.com/2015/10/19/tempest-a-tin-foil-hat-for-your-electronics-and-their-secrets/

critical infrastructures such as military networks or nuclear facilities. In the recent years, several software attacks targeting these systems were disclosed such as *Stuxnet* attack on Iranian nuclear facilities [72] or the *Agent.btz* attack on USA military networks [71].

An air-gapped covert-channel RHA relies on a malware that was initially introduced within an air-gapped network (through social engineering or an infected USB stick). This malware is specifically dedicated to the extraction of the internal computer/network data, it does so through the establishment of a covert-channel with the outside world. The Ben-Gurion University has been studying a wide variety of covert-channels suitable for leaking data from such a device. They used optical emanations from a hard-drive LED in [62], Fan sound in [61] and electromagnetic radiations induced by memory accesses in [60]. These covert-channels were captured at a distance of several meters by an attacker located in another room or outside of the building (using a camera, a radio or an antenna).

More recently, TEMPEST attacks have been shown suitable for remote SCA attacks. In contrary to covert-channels which consider that the attacker has the control of the emitting (infected) device, remote SCA can steal secrets from a target without necessitating any control nor physical access. The *Screaming Channels* attack published in 2018 [22, 21] demonstrated how mixed-signal ICs may leak EM side-channel information at a longer distance than usual. Mixed-signal chips are widely used in IoT and mobile industry. To reduce cost and time-to-market, IC providers implement both analog and digital circuitry on the same silicon die. The digital circuitry usually implements a processor while the analog circuitry implements radio transmitters for Bluetooth and Wifi wireless communication.

In *Screaming Channels*, the researchers observed that the side-channel leakage generated by the digital logic was propagating through the silicon die and interacting with the poorly isolated analog logic. They showed how the digital noise was amplified and transmitted into the air by the chip's analog radio transceivers. The side-channel leakage getting modulated onto the output radio signal could then be retrieved from a distance of 60 meters using a radio receiver. By capturing the EM leakage of an AES algorithm running within an IoT microcontroller at a distance of 15 meters, the researchers were able to conduct a CPA attack and to successfully extract the secret key. The *Screaming Channels* discovery received a great interest from the security community as it relaxed the attack requirements for an SCA (no more direct physical access required). Moreover, with the proliferation of IoT systems these types of mixed-signal chips should prevail in the future. *Screaming Channels* enables far-field EM SCA attacks: from a room to another, from outside a building and could be dangerous even in scenarios where the device is not accessible to the attacker.

2.3.2.2 Timing and Microarchitectural Attacks

The Origins of Timing Attacks

When it comes to security, the time taken to conduct cryptographic operations or verify a password is critical. Indeed, by precisely measuring the duration of a critical operation

Figure 2.7 Famous exploit names in the microarchitectural attack field

one may get information on the data processed. In 1996, Paul Kocher introduced the concept of timing attack on public key algorithms [77]. This attack directly exposed millions of devices to data theft and a quick reaction to build countermeasures was necessary to secure smart cards but also networks and protocols against this new threat. Several countermeasures arose such as constant-time cryptography that could defeat timing attacks by making the operation duration independent from the secret key value.

Remote timing attacks have been introduced in 2003 by Brumley and Boneh [19]. Their work introduced network-based timing attacks by exploiting a vulnerability in the Chinese Remainder Theorem (CRT) RSA algorithm optimization [112]. Thereafter, various implementations of authentication programs and cryptographic functions were shown vulnerable to timing attacks. These exploits usually leveraged precise timing measurement entities available from software such as CPU's cycle counters that can reach the nanosecond precision.

Every hardware or software asset that exposes a timing asymmetry is potentially vulnerable to timing attacks. Nowadays, these problems are gaining renewed attention, especially with the advent of Microarchitectural attacks that exploit timing information leakage in hardware buffers and cache memories.

The Advent of Microarchitectural Attacks

Processor manufacturers are in a constant pursuit of performance improvements. For this reason, modern processors rely on complex hardware mechanisms designed to enhance program execution speed by making data accessible in the least amount of time. A common software performance bottleneck in processors is the memory access latency. In computers running at gigahertz frequencies, this latency can heavily damage performances as it costs several hundred CPU cycles to load or store data. For this reason, processor manufacturers have been designing and integrating fast memories (cache) directly within the ICs. These memory caches can store data physically close to the CPU and thus drastically improve the access latency. In modern systems, two or three levels of cache are usually found: L1, L2 and L3. The cache L1 is present in each CPU core. It is the fastest, it has a latency of 1 cycle and it is usually very small (e.g., 128KB L1 cache in Apple M1[17]). The cache L2 and L3 are slower but bigger (e.g., 12 MB L2 cache in Apple M1) and are often shared between all the processor cores. The optimization brought by cache memory is substantial for ensuring decent performances. However, sharing cache

[17] https://en.wikipedia.org/wiki/Apple_M1

memory has also introduced a major security threat in modern systems: cache timing SCA.

Because several processes may store data and instruction in the shared memory caches, they may interact with other processes by overwriting data belonging to other applications, or having their own data evicted from the cache. Cache memories operate on a first-come-first-served basis. If the data has been evicted from cache, the data would have to be loaded from an outer memory (cache miss) and the loading process will take more time. This information can be used as a side-channel vector to extract information from a process or as a covert-channel vector to exchange information between processes. *Flush+Reload* [144], *Flush+Flush* [58] and *ARMageddon* [84] attacks provide methods to extracts secret keys from cryptographic process using cache SCA attacks. These attacks are particularly concerning for cloud infrastructures as they could bypass the sandbox isolation between VMs which are sharing the same cache memories. Moreover, these attacks have recently been shown feasible in real-world scenarios such as in the Amazon cloud instances [66]. Mitigating these attacks seems hardly feasible since it would require abandoning shared caches and thus redesigning new processors from scratch. However, several software countermeasures have been proposed to thwart these attacks such as flushing the cache before every critical operations or preventing the adversary from accessing timing accurate counters [41]. Even if these countermeasures do not completely mitigate the cache attack threat, they make it more difficult for the attacker to collect accurate time-stamps and thus to extract secrets.

The microarchitectural attacks have also been targeting other CPU performance features such as out-of-order execution, branch prediction and speculative execution. These hardware mechanisms improve program execution speed by predicting the result of conditional statements and pre-loading instructions and data into cache in advance. Branch prediction can be really time saving as conditional statements often lead to the same results. With branch prediction the processor will learn and try to guess what will be the next branch taken by the program and thus compute in advance some instructions (using speculative execution). If the CPU was right, the pre-loaded instruction and data will be kept, if not, they will be discarded. In some programs, the prediction accuracy can reach high rates and lead to significant timing optimizations. However, even if the pre-loaded results are discarded, they may still remain stored in the memory cache for a short period of time. *Spectre*, *Meltdown* and *Foreshadow* attacks [86, 75, 133] exploit these optimization mechanisms along with cache attacks to extract secrets from processes. In *Spectre* and Meltdown, the researchers were able to abuse speculative execution mechanisms in order to read and extract secret data from restricted memory areas. These attacks entirely compromised the hardware isolation between processes and once again exposed multi-user systems to data theft.

Spectre and *Meltdown* paved the way to multiple exploits such as microarchitectural data sampling (MDS) attacks that exploit CPU-internal buffers (Line Fill Buffers, Load Ports, Store Buffers) without having to deal with cache memory. *RIDL* [120] and *Fall-*

out [23] used MDS vulnerabilities to leak private data using buffer side-channels. The *LVI* attack [134] even turned the microarchitectural attack concept into a fault-injection mechanism by modifying the data read by victim enclaves in Intel SGX.

We consider microarchitectural attacks as RHAs as they abuse hardware optimization mechanisms to extract secrets. Moreover, the understanding of microarchitectural mechanisms is essential to comprehend how a complex processor works and must be taken into account when conducting local hardware attacks. It seems that the frontier between the hardware and software security worlds is blurring. With the emergence of complex OS systems, hardware security more than ever needs software expertise to build secure systems and the opposite is also true.

2.3.2.3 Software-based Hardware Attacks

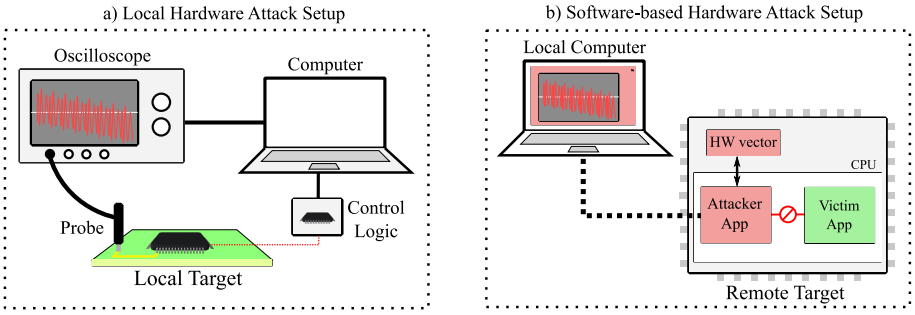


Figure 2.8 Local hardware attack versus software-based hardware attack

This thesis places itself right between the TEMPEST and the microarchitectural attack scenarios. In contrary to microarchitectural attacks that use novel SCA and FIA techniques, we aim to use traditional side-channel and fault injection methods such as power glitch injection and power side-channel to extract data from a device (as in TEMPEST attacks). In contrary to TEMPEST attacks, we want our attacks to be remotely triggered from software (as in microarchitectural attacks). Indeed, we want it to be induced through software so that the attack can be launched on any connected device regardless of the physical distance from the attacker. The widely adopted name for this specific type of attacks is SbHA. We define it as the third type of RHA.

A SbHA can be seen as a traditional hardware attack triggered through software code. In contrary to local hardware attacks, a SbHA does not require any equipment nor physical access to the targeted device (illustrated in Figure 2.8). To conduct a SbHA, the attacker needs to interact with the target's hardware and use this interaction as a fault injection or as a side-channel medium. Before presenting the current SbHA state-of-the-art we introduce the taxonomy employed to better identify and distinguish their modus operandi.

2.3.3 Software-based Hardware Attack Taxonomy

Because SbHAs cover a large spectrum of attack vectors and targeted devices, it seems appropriate to classify them according to different parameters. We identified four major attributes that will help the reader in understanding the major differences between each attack: the vulnerability name, the DUT, the targeted assets and the exploits realized. We describe each parameters in the following paragraphs.

1) Vulnerability Name: As its name suggests, this attribute is usually the name given by the authors to the vulnerability published. SbHA exploits usually come with a short name to facilitate the memorization and maximize the communication around the discovery. We use it to classify the attacks.

2) DUT: The presented exploits have often been tested on at least one target device. Knowing the type of device attacked can be substantial to enable the attack reproduction. In the following classification, we identify the DUT by informing the name of the targeted IC manufacturer or the reference of the DUT.

3) Targeted Assets: This attribute describes the assets that were targeted by the exploit. It may be a victim application running within a processor core, a crypto-accelerator, an enclave, a trusted application, etc.

4) Exploit: The exploit describes which type of attack was conducted on the victim asset. It can be a SCA or a FIA attack depending on the hardware vector used by the attackers.

Additionally to these four parameters that will be displayed in the tables describing each attack, we also discuss three other points in the attack description:

Attack Vector: The attack vector attribute describes the hardware component that was used to induce the hardware attack. In complex ICs, various components surrounding the processor may be used as attack vectors. Despite their differences these components share the characteristic to be accessible from a program running on the processor.

Privileges Required: The amount of privileges required to conduct the attack should be mentioned as it may change depending on the employed attack vector. This attribute is directly linked to the DUT. It may evolve depending on the evaluated target and on the implemented OS.

Reproducibility: Finally, this attribute aims at providing insights to the reader about the reproducibility of the attack. It identifies all the systems that could potentially be endangered by the vulnerability.

In the remainder of this chapter, we provide an up-to-date classification of the published SbHA attacks. The attacks are classified according to the hardware vectors employed for extracting data.

2.3.4 Software-based Fault Injection Attacks

As for local hardware attacks, SbHA can be split into two attack families: Software-based Side-Channel Analysis (SbSCA) and Software-based Fault Injection Attack (SbFIA) attacks. This section describes the use of SbFIA for extracting secrets from remote devices.

2.3.4.1 Overview & Categorization

In 2014, Yoongu Kim et al. discovered a method to induce electrical errors in Dynamic Random-Access Memory (DRAM) memories using only software code [74]. This attack latter called Rowhammer, drastically changed the hardware security field by enabling hardware attack with no equipment and from a remote place.

The significant impact of Rowhammer led to the apparition of various works on SbFIA vectors. At the time of this writing, we identified four different SbFIA methods: Rowhammer, Field-Programmable Gate Array (FPGA)-based SbFIA, Dynamic Voltage and Frequency Scaling (DVFS)-based SbFIA and delay-line-based SbFIA. Each one of them will be categorized in this section with respect to the taxonomy attributes described above.

2.3.4.2 Rowhammer-based Bit-flips Injection in DRAM Memories

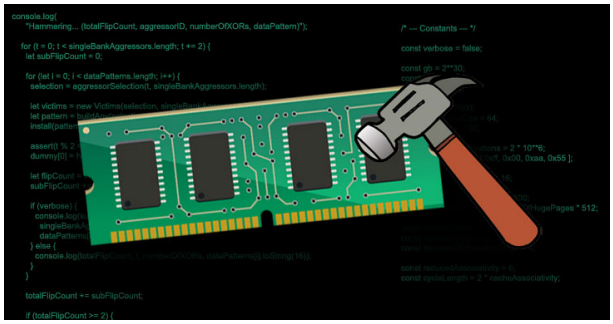


Figure 2.9 Artist's representation of the Rowhammer vulnerability[18]

Background: Rowhammer is a famous vulnerability that arose from the increasing complexity of modern chips. Because DRAM cells density drastically increased in the recent years, it became harder to prevent DRAM cells from interacting electrically with each other. As a result, it was observed that accessing a DRAM location repeatedly (hammering) could disturb surrounding locations and lead to data corruption.

Even if this bug has been known and evaluated by DRAM manufacturers since 2012, its security impact was first demonstrated in 2014 [74]. Kim et al. described a software program that could trigger the Rowhammer bug. The idea is to repeatedly activate two consecutive DRAM rows in order to induce voltage fluctuations and potentially bit flips in a third unaccessed row. This can be achieved using a simple code which continuously

[18] https://fr.techtribune.net/securite/un-nouvel-exploit-javascript-peut-desormais-mener-des-attaques-ddr4-rowhammer/116220/

access data from the two specific DRAM addresses. To make sure that a read/write operation always accesses the DRAM, data caching must be avoided and the memory caches should be evicted for each access. Several instructions such as `clFlush` in Intel processors are used to evict a specific address from the memory cache and enable the attack.

The real challenges behind Rowhammer are to find the appropriate rows suitable for bit flips and a way to exploit the flips for hijacking the system. Methods for finding rows and improving the fault injection rate were proposed in several works [123, 139, 56]. For instance, the introduction of double-sided hammering in [123] yielded to major improvements in bit flip rate and provided additional control on vulnerable DRAM pages. These improvements of the Rowhammer injection mechanism paved the way to various exploits. Table 2.2 compiles a non-exhaustive list of existing Rowhammer attacks.

Table 2.2 Non-Exhaustive list of Rowhammer exploits

Name	**Author**	**Year**	**DUT**	**Targets**	**Exploit**
Flipping Bits in Memory [74]	Kim et al.	2014	Intel, AMD	Linux Process	Random bit flips
Exploiting Rowhammer to Gain Kernel Privileges [123]	Seaborn et al.	2015	/	Linux process	Privilege escalation
Rowhammer.js [57]	Gruss et al.	2015	Intel	Web browsers, Firefox	Privilege escalation
One Bit Flips, One cloud Flops [139]	Xiao et al.	2016	Intel	Xen cloud Platform VMs	Private key extraction, authentication bypass
Flip Feng Shui [115]	Ravazi et al.	2016	Intel	Co-hosted cloud VM	Bit-flip in RSA public keys
Drammer [136]	Van der Veen et al.	2016	ARM	Android Apps	Privilege escalation
Nethammer [87]	Lipp et al.	2020	Intel, ARM	Device in a network	Random bit flips
RAMbleed [83] 	Kwong et al.	2020	Intel	Linux Process	Side-channel on OpenSSH RSA

Attacks: With seven years of existence, the Rowhammer vulnerability has been widely exploited and a large variety of targets have been evaluated. We can distinguish the attacks according to the target and the exploits that were undertaken. Rowhammer is such a powerful threat since it can be launched from a program with limited privileges. For this reason, it was heavily used for privilege escalation. In [123, 57, 136] Rowhammer was conducted from an unprivileged application and employed to gain root privileges. With such rights, an attacker may then be able to access the target hardware, dump the memory and modify the target's functionalities. These privilege escalation attacks were demonstrated feasible on various processors and OS such as ARM and Android in *Drammer* [136] and Linux and Intel in [123, 57]. Even more detrimental, these attacks were demonstrated on DRAM chips using different Double Data Rate (DDR) technologies (DDR3 and DDR4). Because almost all devices embedding a complex OS

use external DRAM memories, the Rowhammer attack could be simultaneously launched on billions of devices. This property is essential for SbHA and particularly fulfilled by the Rowhammer attack.

Other scenarios targeting cloud services have been demonstrated in [115, 139]. They compromised the memory protection in cloud services by enabling unauthorized access to co-hosted VMs. These attacks are particularly frightening for people and companies using cloud services as cloud providers could not be able to ensure proper isolation between users.

More recently, *Nethammer* [87] proposed a method to induce bit flips in a device without requiring local code execution. In this work, the researchers were able to inject bit flips through network requests only and thus drastically reduced the prerequisite for the attacker. In *RAMbleed* [83], the researchers turned the Rowhammer concept into an SCA vector by exploiting the data dependence between Rowhammer-induced bit flips and the bits in nearby rows. This read side-channel was leveraged to extract RSA private keys from victim processes.

Countermeasures: Because definitively thwarting the Rowhammer bug would require the replacement of every DRAM chips deployed on the field, the Rowhammer bug had to be mitigated using resources already available in the targeted systems. For instance, [74] pointed out that increasing the DRAM refresh rate could limit the Rowhammer success rate. Moreover, error correcting codes were often mentioned as suitable countermeasures to detect and correct bit flips. Despite making Rowhammer more difficult, both these countermeasures were defeated few years later [27]. Software countermeasures were also proposed such as *Can't Touch This* [17] which provided a method to separate kernel and user DRAM pages to prevent privilege escalation. However, this countermeasure was defined as unpractical by other Rowhammer-related papers [137, 56]. To conclude, the Rowhammer threat is probably too recent to be properly mitigated and as for the smart cards in the nineties, the race between attackers and security researchers will probably persist over the years.

2.3.4.3 FPGA-based Power Glitch Injection

Background: An FPGA is a digital circuit that can be customized after manufacturing. Historically, FPGAs have been widely used for ASIC prototyping and hardware acceleration applications. They are commonly used along with processors to accelerate certain parts of algorithms in real-time, cryptographic and Artificial Intelligence (AI) applications. The main advantage of an FPGA is it flexibility that makes it possible to build any digital hardware design. They are usually chosen in applications where the production volume is small as the design of a dedicated ASIC would be too expensive. FPGAs are often used in critical applications such as aerospace, defence or security. For this reason, their resistance to local hardware attacks has been widely studied over the years. Notably, the bitstream encryption mechanisms that protects the intellectual property of the design implemented within the FPGA [96, 97, 88, 37].

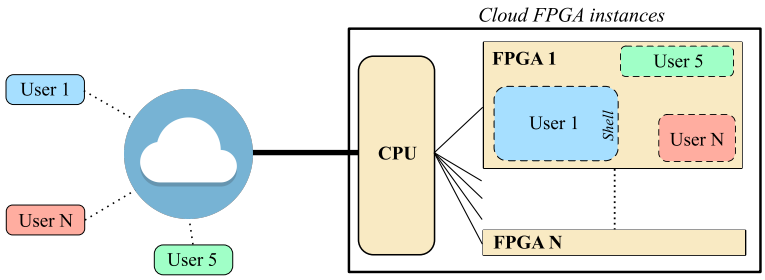


Figure 2.10 Cloud FPGA fabrics shared between multiple users

Because FPGAs can be reprogrammed they became a desirable solution for the industry as AI and security algorithms are evolving quite fast. Moreover, they are now widely adopted in cloud datacenters for acceleration means but also as a service for the end-users. Cloud providers recently deployed FPGA instances in large scale datacenters such as Amazon EC2 [109] and Alibaba F3[19] instances. These services allow users to rent logic resources for big data analytics, AI, security and video processing. Remote access to FPGAs in the cloud raises concerns about associated security threats. The possibility to virtualize and share FPGAs between multiple users was discussed in several design articles and could prevail in the near future [24, 131] (depicted in Figure 2.10). Although logical isolation has been suggested to protect each logic block from the others, recent papers warn the community about the multi-tenant threat. A malicious user could try to take advantage of his configurable resources to eavesdrop or disturb calculations conducted by other users located in the same fabric.

Attacks: Multi-tenant FPGAs can be compared to multi-tenant cloud computers. Instead of having VMs isolating users from each other, the multi-tenant FPGA implements shells to prevent a user from accessing the hardware of another user (Figure 2.10). We've seen in the previous sections on Microarchitural and Rowhammer attacks that the VM isolation in computers could be defeated through RHAs. Several research works have been demonstrating that the FPGA can be equivalently affected by the RHA threat. FPGA-based power glitch injection is a type of SbFIA that may affect cloud FPGAs if a malicious user is in capacity of implementing glitch injection logic. Table 2.3 lists all the FIA methods that have been discovered on FPGAs.

In 2017, Gnad et al. [48] were the first to demonstrate that certain hardware designs could induce voltage drops in FPGAs and lead to faults in computations. Even if, the attack concept is the same as local glitch injection (timing violations in critical paths), the injection mean is now located within the FPGA and is a part of the design. In [48, 80, 16], the injection mean was built using Ring-Oscillator (RO). ROs can be implemented within an FPGA without violating any design rule, they generate oscillations that are latter used to feed either a VCO or a Random Number Generator (RNG). However, it appears that RO oscillations also have a strong influence on the FPGA voltage activity and, by implementing multiple ROs and enabling them at the same time this could lead to voltage glitches.

[19] https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057

Table 2.3 List of FPGA-based Power Glitch Injection exploits

Name	Author	Year	DUT	Targets	Exploit
Voltage drop-based fault attacks [48]	Gnad et al.	2017	Xilinx FPGAs and Zynq	cloud FPGA	DoS
FPGAhammer [80]	Krautter et al.	2018	Xilinx, Intel, Lattice	Multi tenant cloud FPGA	DoS and DFA on AES
RAM-jam [3]	Alam et al.	2019	Xilinx FPGA	Multi tenant cloud FPGA	Bit flips
Neighbors From Hell [16]	Boutros et al.	2020	Intel FPGA	Multi tenant cloud FPGA	Deep learning prediction accuracy degradation

The fact that a valid FPGA design may lead to fault injection or DoS attacks is frightening for the global FPGA security as a malicious Intellectual Property (IP) could embed such a trojan to alter the functionalities of a device. Especially these days where a lot of FPGA IPs are made available by companies or open-source repositories and potentially re-used by designers without performing a security sanity check.

In *FPGAhammer*, an RO-based glitch attack was conducted against a hardware AES implementation. By collecting faulty ciphertexts, the hackers were able to retrieve the AES secret key without accessing the protected shell. In [16], an equivalent attack was conducted on a deep learning hardware algorithm and succeeded in affecting the prediction accuracy.

Countermeasures: Because FPGA-based power glitch injection uses valid bitstreams, it isn't trivial to detect it. A straight-forward mitigation would be to forbid the implementation of ROs in cloud applications. However, this would also prevent users from implementing a large variety of designs that use ROs for valid reasons. Moreover, [3] demonstrated that voltage glitches can be injected through other means such as memory collisions. In [49], it was proposed to build an FPGA antivirus to detect electrical-level attacks. For now we have no information if this idea has been adopted in cloud services or defeated by another attack. As for Rowhammer, the race is far from over.

2.3.4.4 DVFS-based Power/Clock Glitch Injection

Three years after the first Rowhammer exploit in 2014, FPGA-based power glitch injection attack was introduced. The same year, a remote glitch attack was demonstrated on a smartphone platform. Because mobile phones and computers rarely rely on FPGAs, the hackers had to find another method to induce glitches. Here, they used a common power management tool implemented in SoC devices and known as DVFS.

Background: Clock and power management is critical for each electronic device that aims at being power efficient. Smartphones and laptops battery life can be drastically improved by modifying the frequency and voltage of the processor according to the computing demand. The DVFS usually undertakes this task in modern processors. To

that end, it relies on a complex software-hardware cooperation. On the hardware side, the DVFS uses programmable regulators to adjust the processor cores voltage and frequency. On the software side, the DVFS is handled by OS kernel services which continuously update the hardware regulator values with respect to the current processor usage.

In ClkSCREW [128], Tang et al. were the first to raise concerns about the potential security risks exposed by this software-hardware cooperation. First of all, the authors noticed that a privileged untrusted code could access and modify the DVFS calibration. Then, they observed that the available voltage/frequency values were not necessarily bounded to operating limits of the IC and that it could be programmed to inject glitches. The DVFS-based fault injection was then achieved through overclocking or undervolting the processor cores. As for local FIA, various parameters such as glitch strength or glitch width could be precisely calibrated to obtain usable faults. However, the fact that the malware injecting glitches ran within the target involved numerous difficulties linked to noisy OS environment, timing precision, timing resolution, crashes, etc.

Attacks: In contrary to common belief, having root or kernel privileges does not necessarily mean having full control of a device's hardware and memory. As we mentioned in 2.3.1.3, rich OS cannot be trusted and for this reason hardware-based TEEs have been made available by processor manufacturers to build secure applications on top of an insecure system.

Because it accesses hardware registers to inject the fault, DVFS-based fault injection may require root privileges depending on the implemented OS. Therefore, it may be essentially employed to attack higher privilege entities such as HW-TEE (SGX, SEV, TZ) or assets that cannot be directly accessed from the main processor (e.g., secure elements, cryptographic accelerators). Table 2.4 lists all the current publications related to DVFS-based glitch attacks.

Table 2.4 List of DVFS-based power glitch injection exploits

Name	Author	Year	DUT	Targets	Exploit
ClkSCREW [128]	Tang et al.	2017	ARM	TrustZone	self-signed code loading, AES key extraction (DFA)
V0LTpwn [73]	Kenjar et al.	2019	Intel	SGX	OpenSSL SHA
Voltjockey [111] 	Qiu et al.	2019	ARM	TrustZone	DFA on TZ authentication RSA
Plundervolt [100] 	Murdock et al.	2020	Intel	SGX	DFA on AES, Bellcore on RSA

In *ClkSCREW* [128] and *VoltJockey* [111] attacks, ARM processors implementing TZ were targeted (Google Nexus smartphones). In these papers, the attackers abused the fact that DVFS wasn't TZ protected. Thus, a normal world malware was used to inject faults on a secure world AES trusted application by leveraging a granted access to DVFS

tools. By collecting the faulty ciphertexts, the attackers were able to retrieve the AES key and thus demonstrated that the TZ isolation could be compromised using SbHA [128]. Even more detrimental, the hackers were able to breach the TZ authentication system that normally prevents a user from loading custom applications within the TEE [128, 111].

The DVFS-based glitch mechanism was also evaluated on Intel platforms to break SGX enclave isolation. V0LTpwn [73] and *Plundervolt* [100] attacks leveraged hardware-level rights to access *Model-Specific-Registers* and control the processor cores voltage. Similarly to previous ARM exploits on TZ, they were able to defeat the isolation provided by SGX. In *V0LTpwn*, OpenSSL hashes computed in an enclave were successfully faulted. *Plundervolt* went further by retrieving enclave secret keys through a DFA attack on AES and a Bellcore attack RSA-CRT signature.

Countermeasures: In paragraph 2.1.5.1, we mentioned that glitch attacks could be mitigated using detection logic instantiated within the hardware. Because DVFS-based FIA leverages the same injection vector, it should also be affected by such a countermeasure. However, few systems implement these solutions since they are expensive and potentially counterproductive in case of false positive.

Because each processor core may have its own hardware regulator, an alternative solution would be to place the TEE on a dedicated core and prevent the OS kernel from modifying its DVFS calibration [111]. Finally, when a TEE is in use, it seems appropriate to delegate the monitoring of the CPU voltage and frequency to a TEE application. This app could also trigger a warning or a voltage/frequency recovery if the DVFS command gets modified by another process.

2.3.4.5 Delay-Line-based Glitch Injection on Memory Transfers

Today's integrated memory controllers use complex hardware such as delay-lines to monitor and control signal timings during data transfers with an external memory. Because memory chips with different timing specifications may be used, delay-line tuning registers often remain accessible and programmable from the application processor.

During our thesis work, we discovered the concept of delay-line-based fault injection that we used to induce faults in memory transfers and to jeopardize the security of concurrently running assets. This work was named *FaultLine* and as shown in Table 2.5, it is the first of its kind. Chapter 5 of this manuscript is dedicated to the description of this novel SbFIA medium.

Table 2.5 List of delay-line-based glitch injection exploits

Name	Author	Year	DUT	Targets	Exploit
FaultLine [51]	**Gravellier et al.**	2020	ARM	Linux Processes	AES key extraction (DFA and Persistent Fault Attack (PFA)), Bellcore on RSA

2.3.5 Software-based Side-Channel Analysis Attacks

This second part of the SbHA classification addresses the use of SbSCA to conduct on-chip remote power analysis attacks.

2.3.5.1 Overview & Categorization

The main challenge behind SbSCA is to find a hardware component located within the target that can act as a sensor to collect power fluctuations. Once found, the sensor has to meet several requirements to be suitable for SCA. First of all, it has to be sensitive enough to collect small voltage variations induced by a victim asset. Secondly, it should provide a fast sampling frequency to facilitate the identification of the target process and maximize the chances of capturing leakage samples linked to the secret. Finally, because the sensor is integrated within the target, the attacker will face additional challenges such as timing desynchronisation brought by the noisy complex OS environment or privilege right limitations.

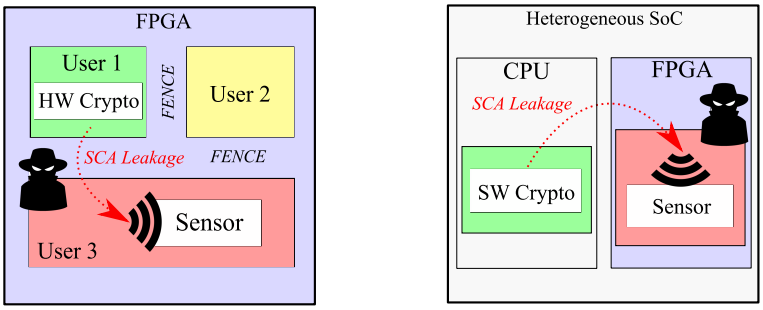
Despite these challenges, the attackers are helped by the increasing complexity of modern devices. A wide variety of components implemented in these SoC devices can be used to capture the power leakage. They were placed here for performance, integrity or security reasons but could be subverted into SCA vectors. Since 2018 and the first SbSCA demonstrated on FPGAs, four SbSCA families have been emerging. The FPGA-based power SCA, the ADC-based power SCA, the delay-line-based power SCA and the Running Average Power Limit (RAPL)-based power SCA. The main difference between these works rely in the vector used for capturing the power leakage. As we did for SbFIA, we will extensively describe each attack in the following sections.

2.3.5.2 FPGAs-based Power Side-Channel Attacks

FPGAs were described in Section 2.3.4.3, they consist in flexible platforms that can be reprogrammed to implement any digital hardware designs. Recently they were massively adopted in the cloud for custom user applications such as AI, video processing or security.

Background: We've seen from an FIA point of view that the apparatus of cloud FPGAs was a bargain for cyber-attackers. In FPGA-based SCA, the objective remain the same: breaking isolation between assets or tenants without accessing them. This time with the help of a propagation delay sensor instead of a glitch injector.

The propagation delay is the time required for a signal to propagate through a logic gate. Power supply, temperature and capacitive effects play a part in the propagation delay equation [35]. While the capacitive load is fixed and temperature can be held relatively stable over the time, voltage fluctuations induce runtime propagation delay variations. At runtime, a sudden under-powering caused by transistors switching activity will induce an increase of the propagation delay throughout the chip. An over-powering will produce the exact opposite. Hence, measuring propagation delays provides an accurate estimation



a) FPGA-to-FPGA Attack *b) FPGA-to-CPU Attack*

Figure 2.11 FPGA-based power side-channel attack threat

of the chip's internal power supply voltage variations. Two major propagation delay sensors are commonly used for FPGA-based power monitoring: the RO-based sensor [150] and the Time-to-Digital Converter (TDC)-based sensor [151]. We describe their principle in chapter 3. Using these sensors, an attacker can eavesdrop the side-channel leakage induced by the overall FPGA computations. By attaching the sensor to a storing mechanism such as a First In First Out (FIFO), he can recreate an oscilloscope, and, if the sensor is accurate enough, conduct SCA attacks throughout the FPGA. Table 2.6 lists some of the existing FPGA-based power SCA exploits.

Attacks: In 2018, Schellenberg et al. demonstrated that FPGA-based sensors were precise enough to be used for SCAs on public and secret key cryptographic algorithms [122]. To enable this attack, the adversary (a TDC-based delay sensor and its control logic for power supply measurement) and the victim (an AES hardware encryption block) had to be located within the same FPGA. This scenario is depicted in Figure 2.11.a. The associated threat model targets multi-user FPGA cloud services that may appear over the next few years. The same year, Zhao et al. disclosed that power SCAs could be conducted on heterogeneous platforms that include both an application processor (CPU) and an FPGA fabric on the same silicon die. This is illustrated in Figure 2.11.b. As a proof of concept, they were able to successfully retrieve the secret key of a custom RSA implementation running within a CPU core [148]. To do so, they carried out an SPA attack using RO-based voltage sensors implemented in the FPGA fabric.

In two works that we conducted in 2019, we introduced new FPGA-based sensors [50] and new attack vectors on heterogeneous SoC, notably CPA on software AES implementations [54]. These findings are described in chapter 3 of this manuscript.

Other works have been evaluating the existence of covert channel mediums in FPGAs [43, 44]. More recently, machine learning accelerators [94] and security assets [45] have been attacked on real-world cloud platforms.

Countermeasures: Because FPGA-based power SCA uses valid bitstreams, it isn't easy to detect. A straight-forward solution to mitigate the threat would be to forbid the implementation of long wires, TDCs and ROs in cloud applications. However, it would also prevent users from implementing a large variety of designs that use these assets for valid reasons. Another way to mitigate FPGA-based SCA in the cloud would be to place

Table 2.6 List of FPGA-based power SCA exploits

Name	Author	Year	DUT	Sensor	Targets	Exploit
An inside job [122]	Schellenberg et al.	2018	Xilinx FPGA	TDC	Multi-tenant cloud FPGA	CPA on AES
FPGA-Based Remote power SCA [148]	Zhao et al.	2018	Xilinx Zynq	RO	Heterogeneous systems	SPA on RSA
Leaky Wires [43]	Giechaskiel et al.	2018	Xilinx FPGA	Long wires	Multi-tenant cloud FPGA	Covert channels
Remote SCA on Heterogeneous SoC [54]	**Gravellier et al.**	2019	Xilinx Zynq	TDC	Multi-tenant cloud FPGA	CPA on SW and HW AES
High-Speed RO-based Sensors for Remote SCA on FPGAs [50]	**Gravellier et al.**	2019	Xilinx Zynq	RO	Multi-tenant cloud FPGA	CPA on HW AES
C3APSULe [44]	Giechaskiel et al.	2020	Xilinx FPGA	RO	Board containing a FPGA	Covert channels
Are cloud FPGAs Really Vulnerable to Power Analysis Attacks? [45]	Glamocanin et al.	2020	AWS EC2 F1	TDC	Multi-tenant cloud FPGA	CPA on AES
Power SCA on BNN Accelerators [94]	Moini et al.	2021	AWS EC2 F1	TDC	Multi-tenant cloud FPGA	Extract image from a BNN accelerator

active fences between FPGA users. In [81], RO fences were used to hide the side-channel leakage of an FPGA AES module. This protection made the SCA more difficult as the attacker had to collect more traces before retrieving the secret key.

To conclude, because there is no easy way to prevent a user from implementing a sensor, the best solution remains in the implementation of side-channel resistant algorithms. This can be achieved by adding masking schemes or clock randomization to temporally spread the side-channel information [147, 149].

2.3.5.3 ADC-based Power Side-Channel Attacks

In 2018, the first SbSCAs were conducted on FPGA. One year later, these attacks were evaluated on microcontrollers. The major difference between FPGA and microcontrollers is that the latter is an ASIC. This means that the hardware cannot be reprogrammed and that the implementation of a TDC or RO-based sensor is not anymore possible. To enable SbSCA in microcontrollers, researchers had to find and use voltage sensing vectors already available within the targeted ICs.

Background: Microcontrollers are small computers convenient for simple tasks that require to be timing reliable and power efficient. These devices are widely used in industry to monitor, control and interconnect facilities but also integrated in various objects to build an interface between the user that can be either physical (e.g., buttons and screen) or dematerialized (e.g., Bluetooth application). Because microcontrollers aim at being used in a large range of applications, they feature a wide variety of hardware peripherals surrounding the main processor (SoC). For instance, a large part of modern microcontrollers embed analog logic for signal sensing, signal generation and connectivity reasons. These

mixed-signal chips (analog & digital) were evaluated as potential target for SbSCA in two 2019 papers as their ADC could be used as a power sensor for SCA. We list these works in Table 2.7.

Table 2.7 List of ADC-based Power SCA exploits

Name	Author	Year	DUT	Targets	Exploit
Leaky Noise [46]	Gnad et al.	2019	STM32, ESP32	IoT devices	CPA on AES
On-Device Power Analysis [104]	O'Flynn et al.	2019	Microchip SAML11	IoT devices, TZ-M	CPA on AES accelerator

Attacks: In *Leaky Noise* [46], it was demonstrated that an attacker with ADC access could monitor on-chip voltage and conduct SbSCA attacks in microcontrollers. By configuring the ADC to probe power pins, researchers were able to collect on-chip voltage fluctuations and store them in memory. Moreover, they introduced a Direct Memory Access (DMA) sampling method to achieve stable and fast ADC-sampling.

In contrary to FPGA in which the signal timings can be exactly calibrated, microcontrollers and computers are subject to timing desynchronization brought by OS and external interruptions. This jitter can be problematic for statistical SCA as multiple traces must be collected to retrieve the key. If the collected traces are not aligned (synchronized), the analysis will fail. The DMA sampling method proposed in [46] solves this problem, by delegating ADC sampling to the DMA which has high priority access to the memory and is not subjected to OS or external interruptions. In the end, the SCA samples are correctly aligned and the sampling rate is improved.

Months later, O'Flynn et al. [104] used ADCs and DMA-sampling to deploy attacks on microcontrollers implementing the recent TZ-M [9]. The TZ-M aims at answering to the growing demand for security in low-cost IoT products by introducing the concept of secure and non-secure world in microcontrollers. TZ-M makes it possible to perform secure operations such as secure boot and secure firmware update by isolating the critical assets and private information from the rest of the application. In their paper, O'Flynn et al. were able to conduct a CPA attack on a microcontroller hardware AES peripheral protected by the TZ-M. This exploit was realistic as the user code could not directly access the AES key. Here, the AES accelerator secret key was betrayed by its power leakage (not taken into account by the TZ-M). The CPA attack required 160 millions traces to extract the entire key from the accelerator. This number is huge compared to the thousand traces required to extract a hardware AES key using TDC-based sensors in an FPGA [50]. This demonstrates the difficulty of achieving SbSCA attacks using only the logic available within a microcontroller. However, these results were promising and led to the discovery of other power SCA mediums that will be presented in the next sections.

Countermeasures: Traditional SCA countermeasures such as masking, shuffling and clock randomization are good candidate to mitigate the ADC-based SCA threat [147, 149]. However, in case of an unprotected algorithm some alternatives are available. In

case of a TZ enabled microcontroller, a simple workaround would be to place the ADC in the secure world and make its use by non-secure world processes impossible [104]. The same could be applied to the DMA mechanism if the ADC had to be kept usable from unprotected processes. Another thwarting method would be to limit the ADC sampling rate or filter its output [46] to prevent it from leaking usable SCA information. Finally, an interesting possibility would be to disable ADC access during secure-world operations.

2.3.5.4 Delay-Line-based Power Side-Channel Attacks

Table 2.8 List of Delay-line-based Power SCA exploits

Name	Author	Year	DUT	Targets	Exploit
SideLine [53] 	**Gravellier et al.**	2020	ARM	Multi-core devices	CPA on AES SPA on RSA covert-channel

During our thesis work, we introduced *SideLine*, a novel power side-channel vector based on delay-line components widely implemented in high-end SoCs. We demonstrated that these entities could be used to conduct remote power SCA and detailed several attack scenarios in which an adversary process located within one processor core aimed at eavesdropping the activity of a victim process located in another core. This work was named *SideLine* and as depicted in Table 2.8, it is the first of its kind. The description of this novel SbSCA medium is done in Chapter 4.

2.3.5.5 Intel RAPL-based Power Side-Channel Attacks

Background: Until the end of 2020, SbSCA remained bounded to ARM devices. The main reason behind this choice was the hegemony of ARM in microcontroller and middle-range SoC systems which were the typical targets evaluated in ADC-based and delay-line-based attacks. However, ARM devices are not the only devices equipped with voltage sensors. They are found in almost all ICs ranging from simple microcontrollers to complex CPUs and GPUs. It was just a matter of time before researchers started evaluating these systems.

Table 2.9 List of RAPL-based power SCA exploits

Name	Author	Year	DUT	Targets	Exploit
Platypus [85] 	Lipp et al	2020	Intel	SGX	CPA on AES, SPA on RSA, Covert-channel

Attack: In the *Platypus* attack [85], Lipp et al. studied Intel processors (Table 2.9). Their research exposed the RAPL interface that can be used to monitor the processor core voltage. It can be used by unprivileged processes to measure the processor power

consumption. Even if, its sampling frequency is extremely limited (max 20 KHz) the researchers were able to use it as a side-channel medium. To that end, they combined various techniques to better control the target execution and its leakage capture. They leveraged cache-based SCA attacks for trace synchronisation, trace aggregation to improve the sensor accuracy and SGX debug tools (SGX-Step) to control the victim execution [135].

Then, this combination of hardware and software expertise was used to conduct power SbSCA on RSA and AES algorithms running within an enclave and a Linux kernel application. Moreover, they introduced novel SbSCA attack paths such as Linux kernel address space derandomization and instruction identification.

Platypus uses a very limited hardware voltage sensing medium but is able to retrieve cryptographic secrets by compensating these restrictions with sophisticated software techniques. SbHAs in general can be perfected by the use of information media directly accessible from the software (e.g., performance counters, cache side-channels, etc). This compensate the SbHA vectors limitations and enable the implementation of combined attacks such as *Platypus*. For this reason, SbHA are and will remain cross-domain attacks, just right in between software and hardware worlds.

Countermeasures: *Platypus* authors proposed several countermeasures to mitigate RAPL-based power SCA. A straightforward method consists in disabling the RAPL via a microcode update. This would prevent both unprivileged and privileged processes from accessing it. Another method would be to make the RAPL interface less precise. For now, the µJ RAPL precision is critical as it is enough to conduct complex SCA attacks. If it was more limited (e.g., mJ), it could remain usable for benchmarking but not for SCA. Practically speaking, the *Platypus* work led to a firmware update on the Linux driver (`powercap`) to prevent unprivileged processes from accessing the RAPL interface. It also led to an Intel microcode update to mitigate instruction identification. These protections will probably be evaluated in a near future.

2.3.6 Software-based Hardware Attack Privileges

When it comes to software-based attacks, the notion of OS privilege rights is substantial to assess the adversary possibilities. In this study, we assume that the SbHA targets are complex SoCs implementing rich OS with multiple privileges levels. To simplify the security architecture of the targeted devices, we split it into three privilege layers: the user mode, the hypervisor mode and the secure mode.

- **User mode**: applications are usually executed in that privilege level. This is the most restricted. In this mode, an application cannot directly access hardware resources. It can only be achieved through userland OS kernel drivers if there are any.

- **Hypervisor mode** (or supervisor mode if there is no hypervisor): it is the highest

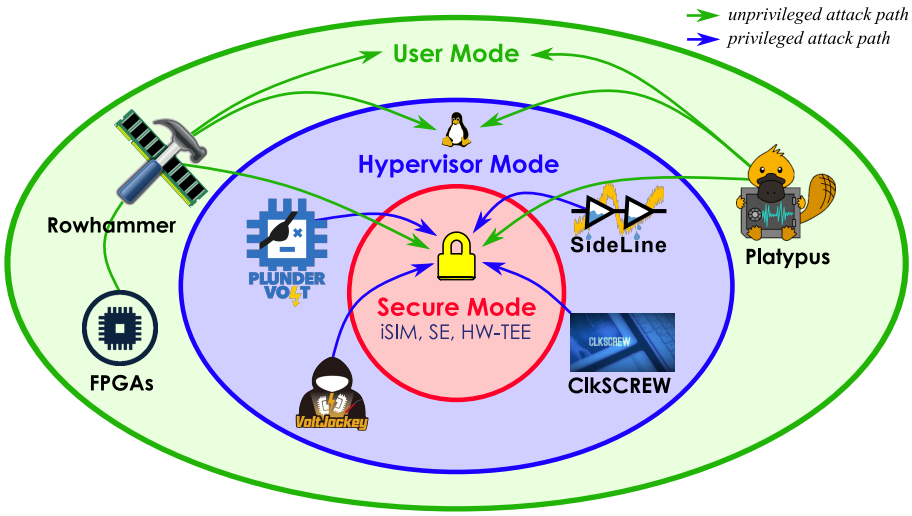


Figure 2.12 SbHA paths and privilege levels

OS privilege level. it allows complete access to privileged instructions and to all the hardware resources that are not protected by the secure mode.

- **Secure mode**: We define the secure mode as the privilege level of all the applications running within secure entities that cannot be directly accessed from the OS. It can be in a hardware-based TEE such as SGX, TZ and SEV or in an integrated secure element (iSIM, key storage unit, secure MCU). The content of these applications is protected from the OS processes regardless of their privilege level.

In this thesis, we distinguish two SbHA paths: The privileged path and the unprivileged path.

The unprivileged attack path illustrated by the arrows in green in Figure 2.12 is directly exploitable through an userland SbHA vector. In this scenario, the malware can target an other unprivileged application, a process running in supervisor mode or any entity executing a program in secure mode. Few SbHAs enable this type of attack because hardware access is often forbidden from in user mode. Therefore, only Rowhammer, *Platypus* and FPGA-based attacks do not require any privileges and can thus target any type of process running within a device. Unprivileged attacks are more likely to spread on remote devices but are also often limited in performances. The lack of tools to synchronize the attack (e.g., hardware performance counters) and the limited interface with the SbHA vector makes it difficult to exploit. For instance, in *Platypus*, the sampling frequency of the RAPL interface was limited to 20 KHz due to the user-space driver latency [85].

An adversary willing to retrieve secrets stored in secure integrated assets would require high SbSCA and SbFIA performances. To that end, he may adopt the privileged attack path depicted with the blue arrows in Figure 2.12. A lot of recent attacks such as ClkSCREW, Plundervolt, VoltJockey and SideLine fall into this category. They leverage supervisor rights to access powerful SbHA vectors and mount secure mode attacks. TZ

and SGX attacks demonstrating that hardware-based TEEs can be defeated by SbHAs have been conducted in various attack papers [128, 111, 100].

In the remainder of this thesis, both attack paths will be evaluated. The attacks reported in Chapter 3 on FPGAs do not require any privileges. The attacks described in Chapter 4 and 5 on SoC devices are likely to require supervisor privileges.

Even if we essentially focused on the identification of SbHA vectors during this thesis, we also tried to build realistic attack scenarios. The attacks reported in Chapter 4 and 5 are usually conducted from an OS application to another. Since the adversary require privileges to access delay-lines the credibility of this scenario is questionable. These attacks could have been reproduced on hardware-based TEEs to improve the realism. However, these entities are not designed to thwart hardware attacks so it is quite likely that our app-to-app attacks are replicable on these platforms. Instead of targeting TEEs, we evaluated secure entities attacks such as CPU-to-Microcontroller Unit (MCU) attacks conducted in Chapter 4. These exploits typically meet real-life scenarios where an application processor delegate security operation to a secure MCU/element that is not accessible from the OS.

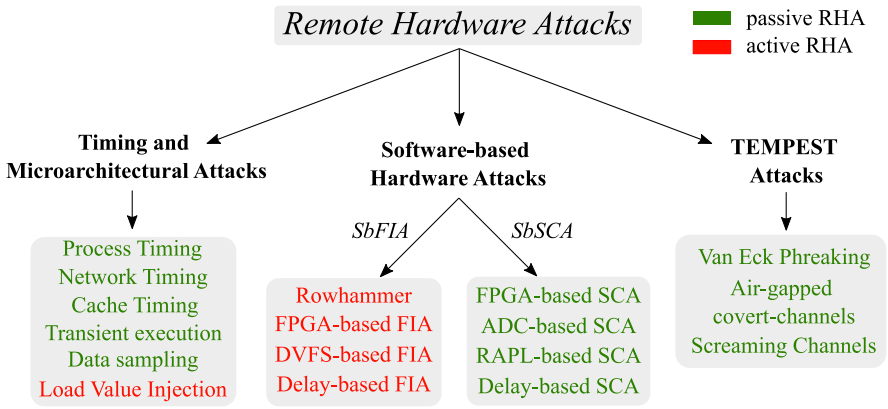


Figure 2.13 Proposed attack classification

2.4 Conclusion

In this chapter, we described the scientific background related to this thesis. More specifically, we characterized the hardware attacks threats and vectors, we described the connected device advent and associated security risks and we classified the RHA types and their applications.

After introducing three types of RHAs (TEMPEST attacks, microarchitectural attacks and SbHA), we exhaustively classified SbHA exploits and provided an up-to-date nomenclature of the threat. Figure 2.13 summarizes the classification work undertaken in this chapter. Each of the three RHA families proposed takes advantage of a hardware vulnerability and its implementation does not require direct physical access to the target. However, these three RHA families also highly differ in the attack vectors used and in the potential threat models.

SbHA is placed in between microarchitectural and TEMPEST attacks. As software attacks, SbHAs can be simultaneously conducted on multiple remote devices connected to a network. However, they essentially exploit hardware vulnerabilities and usually tend to remotely reproduce attacks that are normally conducted locally in hardware security laboratories.

The remaining of this manuscript is entirely dedicated to the study of SbHAs and at the evaluation of their advantages and limitations. The following three chapters describe the SbHA experimentation works conducted during this thesis on FPGA and SoC systems.

Chapter 3. Software-based Power Analysis Attacks on FPGAs

Abstract

In the recent years, Field-Programmable Gate Arrays (FPGAs) have been widely adopted for hardware acceleration purposes in modern systems such as connected devices and cloud datacenters. For flexibility and efficiency reasons, FPGA fabrics are likely to be shared between multiple users in the cloud or implemented along with processors in heterogeneous systems. Despite the logical isolation suggested to protect FPGA tenants or surrounding assets, the generalization of multi-user FPGA environments and heterogeneous systems may lead to novel security threats. Recently, a series of papers demonstrated that a malicious user could use its rented logic in the cloud to conduct remote side-channel and fault attacks on other user assets located inside the fabric or in the surrounding chips. In this chapter, we evaluate this threat and describe various Side-Channel Analysis (SCA) works experimented on FPGAs during this thesis.

Chapter Contents

3 Software-based Power Analysis Attacks on FPGAs **47**

3.1 Chapter Introduction 48

3.2 Technical Background 50

3.3 FPGA-to-FPGA - Designing High-Speed RO-based Sensors for FPGA-based SCA 55

3.4 FPGA-to-CPU - Remote Side-Channel Attacks on Heterogeneous SoC . . . 66

3.5 SCAbox - A Framework for Evaluating the FPGA-based SCA Threat 77

3.6 Conclusion on FPGA-based Power Analysis 84

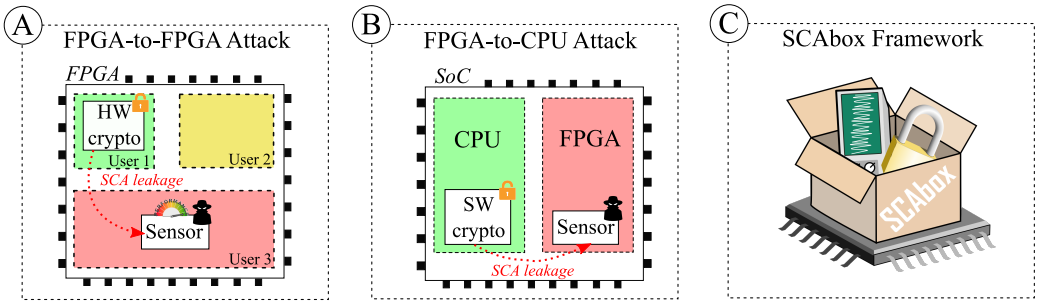


Figure 3.1 Chapter contributions

3.1 Chapter Introduction

In 2018, various types of Software-based Fault Injection Attack (SbFIA) mechanisms had been discovered such as Rowhammer, Dynamic Voltage and Frequency Scaling (DVFS)-based SbFIA and FPGA-based SbFIA. On the Software-based Side-Channel Analysis (SbSCA) side however, the researches were just emerging with the work of Schellenberg et al. In their paper [122], they introduced the concept of FPGA-based power side-channel and its potential risk to multi-user FPGA instances in the cloud. It was rapidly decided that the study of FPGAs would stand as an excellent starting point for this thesis. The main reasons for this choice are described below.

First of all, FPGAs are the most flexible devices available on the market as they allow to build custom digital hardware designs. By working on this type of device, a complete hardware attack setup can be built from the sensor to the target including storage for the SCA traces. Secondly, hardware attacks conducted on FPGAs are not only used as proof of concepts. FPGAs are now widely used as end-products in many application fields and a significant part of them could be threaten by Software-based Hardware Attacks (SbHAs). In particular, datacenters using FPGAs, heterogeneous System-on-Chip (SoC) products and any circuit boards involving FPGAs. The adoption of FPGAs in complex devices for speeding up complex operations related to networks, cryptographic implementations and neural networks seems to get the chip manufacturers interest. Intel, one of the world's largest semiconductor chip manufacturer bought the FPGA provider Altera in 2015 and AMD is said to be in the verge to do the same with the giant Xilinx in 2021[1]. This could rapidly lead to the adoption of FPGAs in Intel/AMD processors and thus in consumer grade computers[2]. Finally, this work on FPGA-based SbHA would act as a transition between local and remote attacks. Indeed, with FPGA-based power SCA we would still have the control on the sensor used and could perform modifications to facilitate the attack. Moreover, the flexibility of the FPGA would be leveraged to evaluate the feasibility of SbSCA applied to different attack scenarios (by varying victim clock speed, noise, attacker sampling frequency, etc).

Our work on FPGA-based power SCA led to the publication of two conference papers

[1] https://www.thestreet.com/investing/amd-acquisition-of-xilinx-approved-by-shareholders
[2] https://www.tomshardware.com/news/amd-patent-shows-CPU-FPGA-integration

and the online release of an open-source framework for facilitating FPGA-based power SCA. These contributions are presented as sections of this chapter.

- Section 3.2 is dedicated to the description of the physical phenomena enabling on-chip voltage monitoring and the presentation of FPGA-based delay sensor implementations.
- Section 3.3 exemplified in Figure 3.1.A describes a new FPGA-based sensor that enables high-speed SCA leakage capture and statistical SCA attacks. This sensor was used to build FPGA-to-FPGA power SCA attacks and thus targets multi-tenant cloud systems.
- Section 3.4 represented in Figure 3.1.B introduces FPGA-based power SCA attacks conducted on heterogeneous SoC systems, that is FPGA-to-Computer Processing Unit (CPU) power SCA attacks.
- Finally, Section 3.5 illustrated in Figure 3.1.C describes a framework dedicated to facilitate the reproduction of the current FPGA-based power SCA works. This open-source toolkit is available on GitHub.

3.2 Technical Background

In this first section, we describe the mechanisms that enable power monitoring in FPGAs using reconfigurable digital logic. We also mention the adopted threat model and describe the existing works already published at the time we began studying FPGA-based power SCA attacks.

3.2.1 FPGAs Voltage Fluctuations

Here, we address the mechanisms that enable FPGA voltage variations monitoring from the origin of power supply fluctuations to their measurement using reconfigurable logic.

3.2.1.1 Power Supply Fluctuations

Power supply fluctuations inside a chip are induced by its transistors switching activity. The quantity of current drawn depends on the resources required for the computation. In complex systems, the voltage is controlled by various components such as Voltage Regulator Module (VRM) that takes part in the Power Delivery Network (PDN).

The PDN has to provide a stable voltage capable to handle power fluctuations. Despite the optimization of the PDNs, transient voltage ripples induced by the interaction between the current drawn and the RLC parasitic component forming the PDN cannot be fully controlled [59, 31]. The parasitic RLC components are depicted in Figure 3.2, they come from the Printed Circuit Board (PCB) tracks, the package bondings and the power grid. These can affect performance and cause timing glitch errors on critical logic paths. Power supply fluctuation leakage through the PDN carries the footprint of the running computation and can be used for side-channel purpose. It can be measured using an oscilloscope connected to the power pads of the target (leading for instance to the waveform illustrated in Figure 3.2) or internally monitored by on-chip sensors that take advantage of its effect on logic propagation delays.

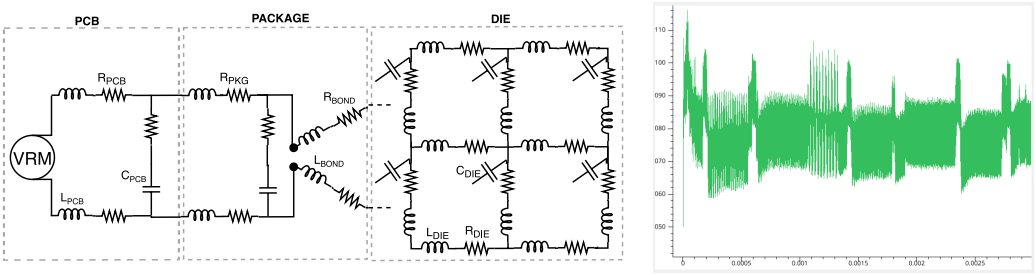


Figure 3.2 RLC parasitic elements in the PDN (left), power supply noise measured on a circuit's power pads (right)

3.2.1.2 Effect on Logic Propagation Delays

The propagation delay is the time required for a signal to propagate through a logic gate. Power supply, temperature and capacitive effects have a significant impact on a logic gate's propagation delay. Equation 3.1 described in [114, 35] defines the propagation time required for a low to high transition tp_{LH} in the case of an inverter logic gate. V_{DD} is the power supply voltage, C_L represents the load capacitance at the output of the gate and V_{th} is the transistor threshold voltage which has a strong relationship with the silicon die temperature.

$$tp_{LH} = \frac{C_L \left[\frac{2|V_{th}|}{V_{DD}-|V_{th}|} + ln(3 - 4\frac{|V_{th}|}{V_{DD}}) \right]}{\mu_p C_{ox} \frac{W_p}{L_p} (V_{DD} - |V_{th}|)} \tag{3.1}$$

While capacitive load is fixed and temperature can be held relatively stable over the time, voltage fluctuations induce runtime propagation delay variations. At runtime, a sudden under-powering (V_{DD} decrease) caused by the switching activity of a target's transistors will induce an increase of the propagation delay throughout the chip. An over-powering (V_{DD} increase) will produce the exact opposite. Hence, measuring propagation delays provides an accurate estimation of the chip's internal power supply voltage.

Figure 3.3 illustrates a waveform view of the propagation delay. When the V_{in} signal controlling the gate switches, it takes tp_{HL} (high to low) or tp_{LH} (low to high) time for the gate to apply changes on the output V_{out}. In the next subsection, we describe how this propagation time tp can be precisely measured using delay sensors.

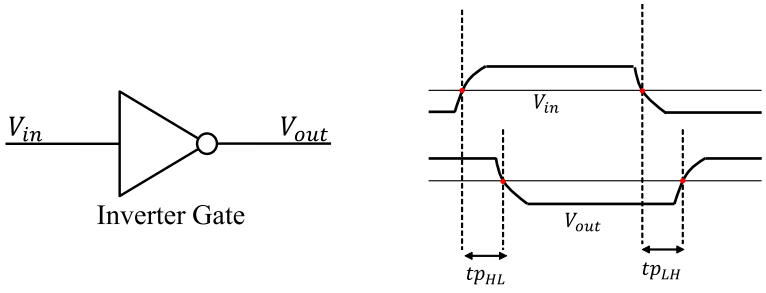


Figure 3.3 Inverter gate propagation delay

3.2.2 Delay Sensors

Two major propagation delay sensors are commonly used for power monitoring in digital circuits: the Ring-Oscillator (RO)-based sensor [150] and the Time-to-Digital Converter (TDC)-based sensor [151]. We describe their principles in the following paragraphs.

3.2.2.1 Time-to-Digital Converter-based Sensor

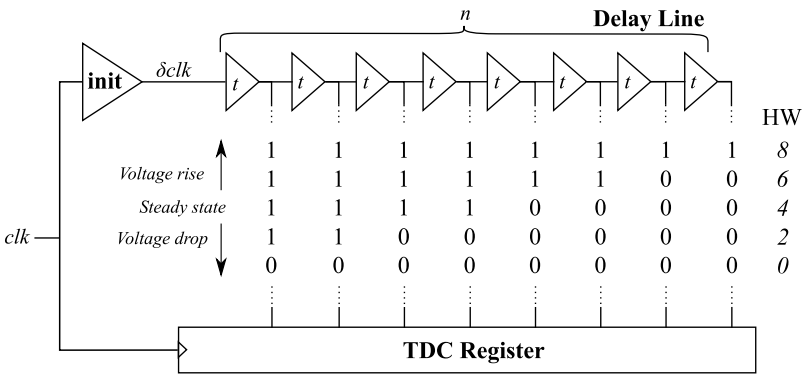


Figure 3.4 Functional schematic of a TDC-based sensor. The Hamming Weight of the delay-line provides an image of the actual on-chip voltage level.

A TDC-based sensor converts propagation delay variations induced by power supply fluctuations into digital information. Thanks to a low-cost design and a fine-grained resolution it is commonly adopted as on-chip temperature or voltage sensor: for operating control of a chip [132] as well as glitch attack detection [151]. More recently, with the emergence of FPGA cloud services, some researchers started to use it to perform power SCA attacks [122, 121]. As it offers a good trade-off between achievable resolution, accuracy and sampling frequency [47, 151], TDC was commonly adopted for SCA experiments. As illustrated in Figure 3.4, a TDC-based sensor contains three main logic blocks:

- An **init** delay block whose propagation delay depends of the chip's internal voltage.
- A **delay-line** made of n elementary delay elements (with an individual propagation delay t) that allows a fine measurement of propagation delay fluctuations.
- A **register** that captures and stores the delay-line state.

A clock signal, denoted clk, is connected to the TDC $init$ delay block input and delayed to form a δclk signal. The phase shift between clk and δclk signals fluctuates with the voltage variations. The $init$ delay is calibrated in order to have the δclk edge inside the delay-line when its state is captured by the TDC register. Then, the Hamming Weight of the stored value is computed and delivers an image of the actual voltage level inside the chip (as a thermometer code). Figure 3.4 illustrates the impact of voltage fluctuations on the sampled value. A voltage rise reduces the propagation delay of the $init$ block. Therefore, the δclk rising edge travels faster and manages to pass more elements in the delay-line. Therefore, more "1" are sampled and the Hamming Weight of the TDC register increases. A voltage drop induces the opposite behavior by increasing the propagation delay and thus, the number of "0". To enable fine-grained voltage sensing, the propagation delay t of the logic primitives constituting the chain needs to be as small as possible. However, a small propagation delay involves a long delay-line to avoid saturation of the

TDC. Therefore, there is a trade-off between quantum resolution, voltage range and area overhead.

3.2.2.2 Ring-Oscillator-based Sensor

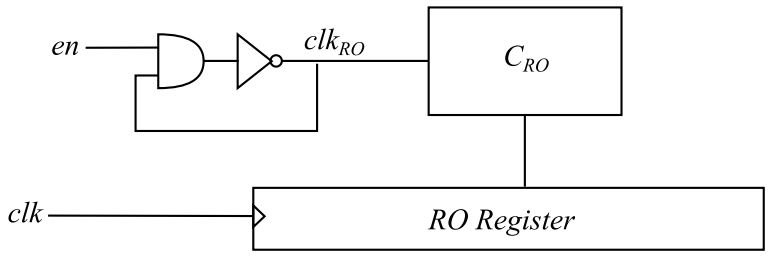


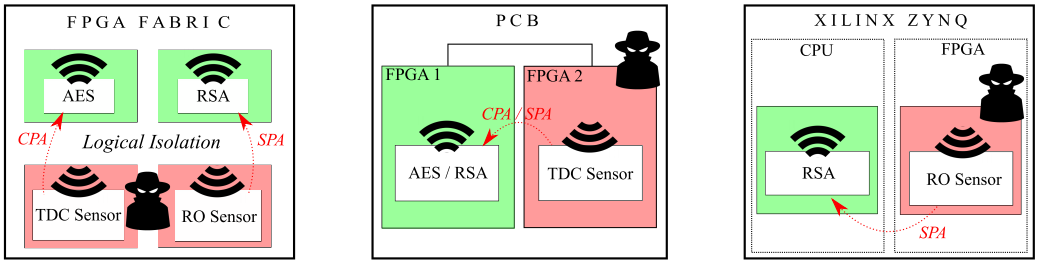
Figure 3.5 Functional schematic of a RO-based sensor

The RO-based delay sensor illustrated in Figure 3.5 monitors propagation delay fluctuations through the measurement of its RO oscillation frequency f_{RO}. A RO is a device composed of an odd number of cascaded inverters (the RO represented in Figure 3.5 has only one inverter gate). The output of the last inverter is fed back to the first creating an infinite oscillation between two voltage levels. The oscillation frequency f_{RO} is defined by the number n of inverters in the ring, each inverter slows down the oscillation because of its internal propagation delay t_p. This results in the following RO frequency equation: $f_{RO} = \frac{1}{2t_p n}$. Therefore, measuring the frequency variations of the RO indicates the propagation time fluctuations of its inverters. Hence, it provides an image of the chip power supply consumption. To enable the measurement of the RO oscillation frequency, designers commonly adopt digital counters [148, 89]. A counter C_{RO} is connected to the RO output: it is incremented by the RO oscillations and it is read out by a register at a fixed sampling frequency f_s. ΔC_{RO} represents the number of RO oscillations counted during a period. The equation that converts the counter value into the RO frequency f_{RO} is:

$$f_{RO}(t) = [\underbrace{C_{RO}(t) - C_{RO}(t-1) + \epsilon}_{\Delta C_{RO}}] * f_s \tag{3.2}$$

3.2.3 Threat Model: FPGA-based Power SCA

Our threat model addresses all the connected devices that incorporate hardware acceleration based on FPGA logic: from reconfigurable resources in cloud data centers to FPGAs deployed for industrial and commercial purposes. In this context, we consider the potential implementation of malicious FPGA-based delay sensors through cloud FPGA rental, malicious Intellectual Property (IP) insertion or access to the bitstream reconfiguration of unsecured chips. Using these sensors, an attacker could eavesdrop the side-channel leakage induced by surrounding computations. In a cloud scenario, a malicious user could capture the side-channel leakage of cryptographic computations conducted by other users.



a) Intra-FPGA Attack *b) Inter-Chip Attack* *c) Heterogeneous-Chip Attack*

Figure 3.6 Overview of FPGA-based Power Side-Channel Exploits

In SoC context, these sensors can be implemented to eavesdrop the side-channel leakage induced by the SoC surrounding logic blocks such as CPU cores [148]. The following sections evaluate FPGAs and heterogeneous SoCs that provide both CPU and FPGA logic blocks within the same die. Our goal in this chapter is to assess the feasibility of FPGA-based SCA attacks on hardware and software crypto-algorithms.

3.2.4 Related Works: Existing Scenarios (2018)

Several works studying these threats were conducted before we started evaluating FPGA-based power SCA. Although being all based on FPGA sensors they introduce three different scenarios as illustrated in Figure 3.6.

1) Intra-FPGA Attack: Remote SCA attacks on FPGAs were introduced in 2018 [122]. The adversary model consists in an FPGA fabric shared among multiple users. Each user is protected from the others by logical isolation. Despite this protection, a malicious user can implement voltage sensors in his rented logic to monitor voltage fluctuations induced by surrounding computations. Assuming this model, the adversary is able to perform a Correlation Power Analysis (CPA) attack against a victim Advanced Encryption Standard (AES) hardware module. In [148], the researchers used RO-based sensors to perform intra-chip Simple Power Analysis (SPA) against an Rivest–Shamir–Adleman (RSA) cryptographic hardware module.

2) Inter-Chip Attack: The inter-chip SCA Attack illustrated in Figure 3.6.b went a step further by proving that an untrusted chip soldered on a PCB could sense voltage variations induced by other chips through the PDN of the PCB. In this exploit, an adversary FPGA was able to perform a CPA attack against an AES module and an SPA attack against a RSA module running on another FPGA fabric [121].

3) Heterogeneous Chip Attack: Heterogeneous FPGA technology integrates both processors and FPGA fabrics within the same SoC. In [148], malicious ROs were implemented in the FPGA fabric to perform an SPA against a naive square and multiply RSA algorithm running on a Linux OS in the CPU core as shown in Figure 3.6.c.

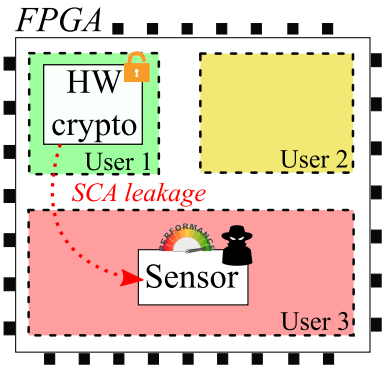


Figure 3.7 FPGA-to-FPGA attack

3.3 FPGA-to-FPGA - Designing High-Speed RO-based Sensors for FPGA-based SCA

The work described in this section was published in [50] with co-authors Jean-Max Dutertre, Yannick Teglia and Philippe Loubet Moundi.

3.3.1 Introduction

The research work conducted in this section aims at improving the existing sensor designs deployed to monitor power supply fluctuations within FPGAs and to push forward the state-of-the-art on FPGA-based SCA attacks. The major contributions of this section are detailed below:

- We propose a new design approach for RO-based sensors that enables nanosecond scale measurement of FPGA internal voltage: the Johnson Ring-Oscillator (JRO)-based sensor.
- We evaluate the usage of the JRO sensor in a multi-user FPGA scenario and experimentally demonstrates its ability to conduct CPA attacks against an AES hardware module.
- We compare different FPGA-based sensor designs (JRO-based sensors and TDC-based sensors) with traditional side-channel methods (ElectroMagnetic (EM) side-channel).
- We demonstrate that, despite the modest quantification level and sampling frequency achievable using FPGA-based sensors, proximity, flexibility and advanced configuration allow them to provide similar SCA results to traditional measurement setups.

In Subsection 3.3.3 we introduce the proposed JRO-based sensor. The experimental setup and the obtained CPA results are described in Subsection 3.3.4. Then, Subsection 3.3.5 establishes a comparison between FPGA-based sensors and traditional EM side-channel. Subsection 3.3.6 concludes this work.

3.3.2 Motivation

Because of limitations in their achievable resolution and sampling frequency, RO-based sensors were at first only used to carry out SPA attacks [148]. This section provides a new design method for RO-based sensors that improves their performances and enable their use for statistical SCA attacks on symmetric encryption algorithms. The adopted approach is to perform an intra-FPGA CPA attack as previously achieved with TDC-based sensors in [122]. The victim shell contains an AES module which continually encrypts data. The power consumption leakage resulting from each AES encryption is acquired by the adversary shell and later used for external CPA computations and AES key retrieval.

3.3.3 A Novel RO-based Sensor Design

This subsection introduces the proposed JRO-based sensor. To begin with, we address the limitations that attackers encountered when using traditional RO-based sensors.

3.3.3.1 RO-based Sensors Downsides

Several RO-based sensors downsides recently pushed researchers to adopt TDC-based sensors for side-channel purposes instead [151, 47]. This mainly comes from the fact that the RO-based sensors struggle to provide reliable measurements when they reach the MHz sampling rate [89, 148, 151]. This limitation comes from three major factors:

1) Frequency Dependant Resolution: The resolution of the RO-based sensor relies on the number of oscillations counted during a sampling period. When a long-sampling period is adopted, a large number of oscillations is counted and a fine-grained image of the voltage level can be retrieved through the capture of the counter value. A decrease of the sampling period reduces the number of RO oscillations counted and limits the relationship between the counter value and the actual voltage level. Therefore, decreasing the sampling period gradually deteriorates the sensor resolution.

2) Quantization Error: The quantization error ϵ (see Eq. 3.2) is a distortion of the RO-counter value by 1 that sometimes occurs because of the absence of a phase relationship between f_{RO} and f_s [148]. At high sampling frequencies the quantization error becomes significant over the total number of RO counter increments. Thus, its occurrence can skew the overall measurement.

3) Counter Timing Error: To enable high sampling frequency measurements, the RO should oscillate as fast as possible. However, at GHz frequency range, timing errors occur if the counter fed by the RO is not optimized for high frequency transitions. This results in the sampling of inconsistent counter values that further alter the RO-based sensor reliability. This counter limitation is discussed in [151, 47] but neither improvements nor new designs were suggested.

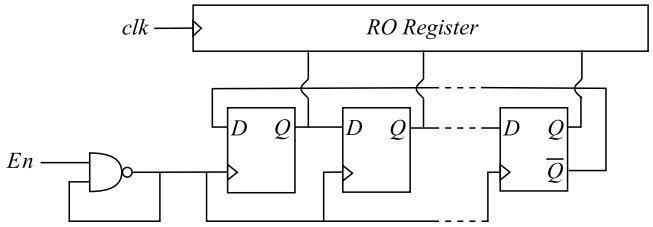


Figure 3.8 Schematic of the proposed JRO-based sensor design. The RO consists in a looped NAND which cadences the JRC. A register reads out the JRC at a fixed rate defined by clk.

Resource	Number	Usage
LUT	2	*RO (nand) & JRC (not)*
DFF (FDCE)	8	*Johnson Counter*
DFF (FDCE)	8	*Sampling Register*
Slices	**2**	

Table 3.1 Resource utilization for 1 RO-based sensor instance.

3.3.3.2 Designing a high frequency RO-based Sensor

The JRO-based sensor design presented in this section is depicted in Figure 3.8. It still consists in three blocks: a RO, a counter and a sampling register. By introducing this sensor, we aim at mitigating the impact of the three main RO-based sensor limitations detailed above.

1) A Faster RO: Because we are working with two clock sources f_{RO} and f_s, our design will suffer from phase shift quantization error. To mitigate this effect and to increase the resolution of the JRO sensor, we implement the fastest RO achievable with the available logic. It consists in only one LUT configured to perform a NAND operation whose output is fed back to its input (see Figure 3.8). The resulting oscillation frequency approximately reaches 1.2 GHz.

2) An Optimized Counter: To preserve the JRO from timing errors caused by the RO speed, we choose a non-binary counter known as Johnson Ring-Counter (JRC)[3] which only consists in cascaded D flip-flops (DFF) associated to an inverter gate. By adopting a design that almost doesn't require combinational logic to be inserted between DFFs, we mitigate timing errors that binary counters would encounter when cadenced by GHz range signals. Our sensor structure is depicted in Figure 3.8. The clock input of each flip-flop is connected to the output of the RO. The data path consists in a ring in which the complementary output of the last flip-flop $\overline{Q}$ (inverter gate) is fed back to the data input D of the first one. Using 8 flip-flops the JRC provides 16 distinct states which is enough when the RO register sampling period is smaller than 16 times the RO period.

[3] https://www.electronics-tutorials.ws/sequential/seq_6.html

The JRO is implemented using Xilinx low-level primitive templates. This ensures that the number of logic gates instantiated in our VHDL source file meets the number of logic gates used in the fabric [140]. Therefore, we make sure that no additional logic can alter the timing margins of our sensor. Contrariwise, hidden combinational gates can be found in various RO-based sensor designs. For example in [148], the counter is made of toggle flip-flops and appears free from combinational gates. However, toggle flip-flop does not exist as a primitive in the fabric [140] and should be constructed using combinational and sequential logic (e.g., DFF + XOR). Hence, in this previous RO-based sensors implementation, the combinational delay problem persists.

3) A Lighter Design: A JRO-based sensor instance consumes only 2 slices as detailed in table 3.1. Such a small design can be spread throughout the fabric without area congestion. Thus, the area coverage and more importantly the overall resolution of the voltage sensor can be improved. This statement is discussed in the following part.

3.3.3.3 Number of RO-based sensors

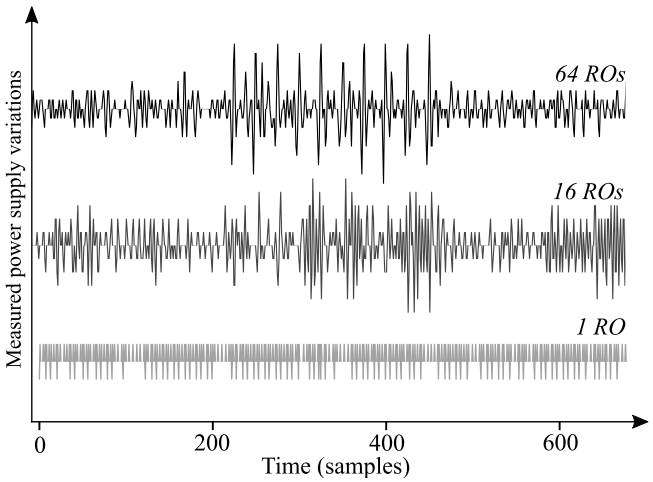


Figure 3.9 Effect of the number of JRO-based sensors on the overall resolution. The depicted signal is a single trace of an AES encryption running at 10 MHz.

Equation (3.3) expresses the counter value, ΔC_{RO}, as a function of the sampling frequency f_s and the RO frequency f_{RO}. The RO frequency f_{RO} is split in two terms: a constant one $f_{RO(natural)}$ that represents the steady state natural oscillating frequency of the RO and a dynamic one $f_{RO(dynamic)}$ which depends on the activity of the surrounding logic.

$$\Delta C_{RO} = \frac{f_{RO(natural)} + f_{RO(dynamic)}}{f_s} \tag{3.3}$$

When several sensors are instantiated within the fabric, their contribution ΔC_{RO} is summed and averaged over the number of JRO-based sensors used. Multiplying the number of JRO-based sensors throughout the chip has several benefits. Firstly, because of process and routing paths variations, each JRO has a specific phase and frequency. For

this reason, the quantization error ϵ only affects a portion of the sensors simultaneously. When the number of JRO-based sensor used increases, the quantization error progressively loses significance over the global voltage fluctuation measurement. Therefore, it has less impact regarding the accuracy of the overall sensor. Secondly, the natural frequency deviation between each JRO instance enhances the granularity of our sensor. Depending on the value of $f_{RO(natural)}$, the dynamic frequency fluctuation $f_{RO(dynamic)}$ required to modify the counter value fluctuates. Therefore, each JRO-based sensor instance provides a specific contribution that further enriches the overall resolution.

Figure 3.9 illustrates the effect of the number of JRO-instances on the sensor resolution (*our experimental setup will be described in Subsection 3.3.4.1*). A single 10 MHz AES encryption was captured using 1, 16 and 64 JRO-based sensors cadenced at a **250 MHz sampling rate**. This is 125 times faster than previous RO-based sensors used in [148]. When only 1 JRO is used, the quantization error effect is maximal and the $f_{RO(dynamic)}$ fluctuation only provides 3 distinct quantization levels (Figure 3.9 - *1 RO*). Thereby, the AES encryption is not visible in the obtained waveform. However, increasing the number of JROs gradually leads to the appearance of the AES activity over the residual and quantification noises. Using 64 JROs the 10 AES rounds are clearly visible.

3.3.3.4 Place and Route Influence

Manual place and route is not required to enable JRO-based voltage fluctuation measurement. However, when it is possible, designers can set the placement and routing paths using relative placement macro (RPM) to improve JRO-based sensor performances and get a better control of their distribution throughout the fabric.

3.3.4 RO-sensor based Correlation Power analysis Attack

The following subsection provides results of a CPA attack conducted using the JRO-based sensors on an hardware AES module implemented within the FPGA fabric.

3.3.4.1 Experimental Setup

A Xilinx Zynq SoC that provides both CPU and FPGA on the same die was adopted for our experiments (note that the CPU was not targeted in this section, we exclusively focused on an intra-FPGA exploit).

Our experimental setup is depicted in Figure 3.10. From the victim point of view, the CPU was dedicated to the management of the AES plain and ciphertexts while the attacker program was developed for sensor calibration and measurement exportation. The FPGA fabric was separated in two logically isolated blocks with distinct clock regions. Because in a multi-user scenario the victim shell may not be necessarily placed next to the adversary, we instantiated the victim AES as far as possible from our sensor instances. Hence, we aimed at demonstrating the resilience of our sensors to their distance from

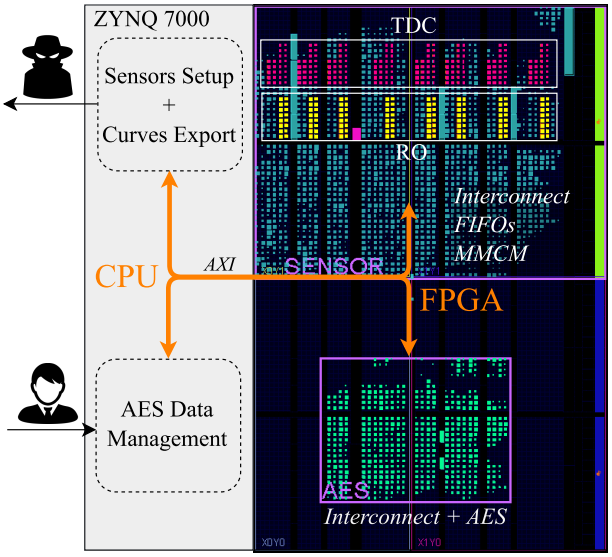


Figure 3.10 Xilinx Zynq Multi-User Experimental Setup.

the target and to the noise caused by the surrounding logic. The adversary shell contained 64 JRO-based sensors (128 slices) and 8 TDC-based sensors (208 slices). The remaining logic was dedicated to Advanced eXtensible Interface (AXI) bus interconnect, First In First Out (FIFO) and clock management. Please note that a big part of the logic was implemented for experiment purposes and could have been removed to get a lighter implementation.

The hardware AES module instantiated in the victim shell (in green) was dedicated to the encryption of sensitive data. It loaded 128-bit packets of plaintexts from the CPU using shared AXI registers, encrypted them and returned the computed ciphertexts. It relied on a 128-bit secret key and each round was executed in one clock cycle at 50 MHz. With 1,500 LUTs and 400 FFs, this AES module consumed around 10% of the total fabric.

3.3.4.2 Correlation Power Analysis Model

A power SCA attack relies on the fact that CMOS power consumption leakage depends on the handled data. By writing down a model of the expected AES power consumption and combining it to the sensor voltage measurement, a CPA attack as described in [18] should allow us to retrieve the AES secret key. The attack conducted in this section targets a state register that temporarily stores data resulting from each AES round transformation from the plaintext importation to the ciphertext generation. This 128-bit register is synchronously refreshed at the end of each round generating a strong switching leakage that significantly affects the power supply level. The leakage level resulting from the register update fluctuates according to the Hamming Distance between the previous and the current AES state. Targeting this register requires the knowledge of two consecutive states. In our case, we assumed that the adversary had access to the ciphertext

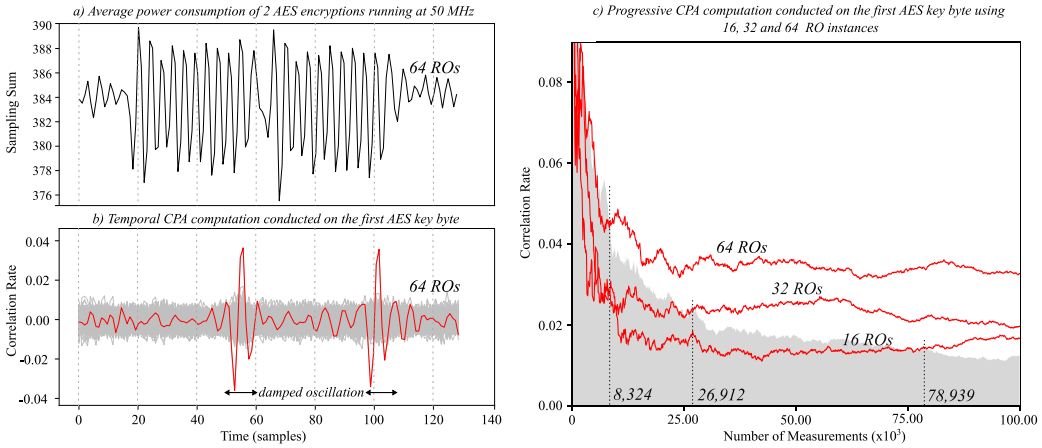


Figure 3.11 Averaged AES power consumption (a) and CPA results (b)(c) by means of 100,000 traces acquired using 16, 32 and 64 JRO-based sensors. The right key hypothesis candidate is represented in red in (b) and (c).

which is also the last value stored by the AES state register. We adopted the last round attack model described in [92] and computed the correlation rate between the model and the experimental curves. If the adopted model is relevant, the correlation rate of one of the key hypotheses “right candidate" should be distinguishable from the others “wrong candidates".

3.3.4.3 JRO-based Sensor CPA Results

The power consumption resulting from the AES encryption was acquired 100,000 times using the 64 JRO-based sensors. The measured average power consumption is represented in Figure 3.11.a (two successive encryptions are depicted but it has no effect on the side-channel results). Several experiments were conducted to evaluate the CPA results provided by the JRO-based sensors.

1) Number of sensors: Figure 3.11.c shows the CPA results obtained using different numbers of JRO-based sensors. Using 16 JROs, it took around 79,000 traces for the right candidate to emerge from the wrong key hypotheses. With 32 and 64 JRO-based sensors the number of required traces dropped to 27,000 and 8,000 respectively. This attack was also conducted using only 1 JRO-based sensor but required almost 1 million encryptions. In conclusion, the number of required traces to infer the secret key appeared to be inversely proportional to the number of JRO-based sensors used.

2) Target frequency: In order to study the impact of the target speed on the CPA results, we conducted the same experiment for different AES frequencies from 10 to 200 MHz. However, increasing it did not significantly changed the CPA results. To explain this phenomenon, we investigated the Zynq response to transient voltage fluctuations. Based on the results obtained, it appears that when a sudden voltage drop occurs within the chip (e.g., update of the AES state register), the parasitic capacitive and inductive elements forming the PDN resonate and the voltage level temporarily oscillates

until finally reaching its steady-state value [132, 31]. The damped oscillation induced by the 10th round update of the AES state register can be seen in temporal correlation results depicted in Figure 3.11.b (f_{aes} = 50 MHz). As the voltage transient response is fixed by the parasitic components forming the PDN, the amount of time during which the side-channel leakage can be leveraged is not bounded to the AES frequency but to the device itself [129]. Actually, we observed the exact same oscillation effect for each AES frequency. This experiment demonstrates that, for side-channel purposes, the sensor sampling frequency can be lower than the victim operating frequency. However, it has to be high enough to ensure that the victim's valuable side-channel leakage is properly sampled. Through the Zynq transient response oscillation frequency measurement ($\approx$ 50 MHz), we get an idea of the sampling frequency required to successfully perform the attack (Nyquist-Shannon sampling theorem: $f_s > 2f_{leak}$). Thanks to the 250 MHz sampling rate offered by the JRO-based sensors we were able to accurately retrieve the side-channel information.

3.3.5 Further Results and Discussion

For comparison purpose, this subsection goes further by adding side-channel results acquired using TDC-based sensors and a traditional EM side-channel setup. We evaluate the performance of each configuration and discuss about use cases and countermeasures for on-chip sensors.

3.3.5.1 TDC & Electromagnetic Experimental Setup

The adopted TDC-based sensors provided 32 quantization levels and a sampling rate of 250 MHz. Each instance consumed 26 slices and 8 of them were implemented within the fabric (illustrated in Figure 3.10). The calibration of the TDC delay-line had to be done manually by modifying the number of logic elements forming the $init$ block. Each TDC was initialized independently before the measurements.

The EM setup consisted in a near field microprobe connected to an oscilloscope with a 5GS/s sampling rate and a 12-bit resolution. The probe position was controlled using a X, Y, Z table. The signal was first amplified by a low noise amplifier before being fed into the oscilloscope. The electromagnetic leakage of the first AES round was used to trigger the oscilloscope. The captured samples were then extracted and used to perform a Correlation ElectroMagnetic Analysis (CEMA) [40].

3.3.5.2 Side-Channel Results

A single campaign of encryption in which the three measurement setups simultaneously acquired the side-channel leakage was conducted. Figure 3.12 presents a bar chart showing the number of traces needed for the right guess to emerge depending on the measurement setup. Results are given for the first 8 bytes of the AES encryption key. TDC-based

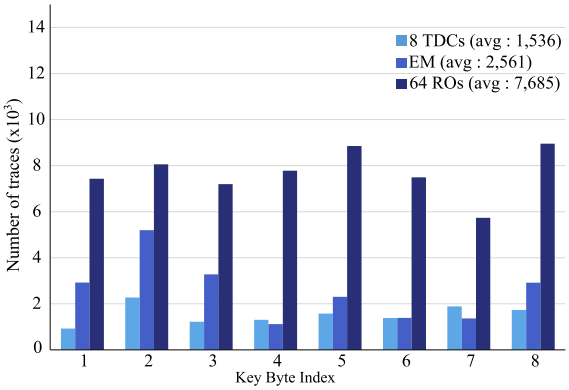


Figure 3.12 Number of traces required for the right candidate to emerge from the wrong key hypotheses. Results are given for 8 bytes of the encryption key using 3 measurement setups: EM, 8 TDC-based sensors and 64 JRO-based sensors

sensors provided results comparable to that of the EM setup while JRO-based sensor remained 3 or 4 times less efficient than the other setups. Despite a significant difference of sampling frequency and resolution between integrated sensors and oscilloscope, the obtained results are quite similar. Naturally, these results must be interpreted with caution as TDCs and JROs were previously calibrated and optimized for this specific device and attack scenario. Our JRO-based sensors do not reach the level of accuracy of TDC-based sensors but are still precise enough to successfully perform a CPA. Moreover, they benefit from significant implementation advantages that will be addressed in the following subsection.

3.3.5.3 Discussion

1) On-chip sensors comparison: When designing on-chip voltage sensors a trade-off needs to be made between achievable resolution, sampling frequency and area coverage. Depending on the use case, a sensor will be more relevant than the others. Regarding the results of CPA conducted in this section, our novel JRO-based sensor remains slightly less efficient than TDC-based sensors for side-channel purposes. However, this sensor offers a better flexibility and scalability than TDC-based sensors. Thanks to their light implementation JRO-based sensors can be spread through all the chip without congestion. Thus, they provide a better coverage of the power supply voltage fluctuations throughout the fabric with a lower area cost. Moreover, they don't need any calibration or specific logic cells contrarily to the TDC-based sensor which requires an init delay configuration to control the position of the clock edge inside the delay-line as well as specific CARRY4 logic to provide reliable measurements[4]. This suggests that JRO-based sensors would be easier to transpose on devices integrating different manufacturing processes.

Finally it is important to remember that previous RO-based sensor designs were never used for conducting such statistical CPA attacks as their sampling frequencies were lim-

[4]The CARRY4 logic gates can be interconnected without requiring access to the FPGA switch matrix. This ensures a stable propagation time of between cascaded CARRY4 logic gates and improve the TDC delay-line stability.

ited to KHz [150] and 2 MHz in [148] against 250 MHz for our JRO setup. The RO-based sensor SCA characteristics can be highly improved by adopting the proposed JRO design.

2) Potential use-cases: RO-based sensors have been widely adopted for voltage and temperature monitoring [150], attack detection [89] and more recently SCA attacks [148]. The JRO-based sensor described in this section was shown suitable for statistical SCAs and could be used to improve the RO applications mentioned above. A further side-channel use case for on-chip sensors was discussed in [43] and consists in their implementation as hardware Trojans. FPGA end-products often include third-party IP blocks because of the high-cost of design and development. Considering the critical application in which FPGAs are deployed, the potential integration of FPGA-based trojans through untrusted IPs could lead to disastrous consequences. (e.g., industrial espionage, denial-of-service, etc).

3) Side-channel countermeasures: Multi-user FPGAs haven't been launched yet but several technical papers already shown the multi-user feasibility and the benefits that a highly scalable and flexible multi-user service could provide to tenants and more specifically to cloud providers. Despite the fact that logical isolation between logic blocks is ineffective against power SCA attacks [152], side-channel threats could be easily mitigated by restricting manual place and route and forbidding combinational loops that enable the RO implementation. The problem lies in the fact that these features are essential for a lot of FPGA applications and their suppression would significantly alter the service. Trojan detection routines could also be developed to prevent designers from implementing on-chip sensors but would require a lot of developments and would be soon challenged by novel adversary designs bypassing the security. At the moment, there is no easy way to mitigate FPGA hardware attacks other than preventing multi-user FPGA systems.

3.3.6 Conclusion

This section introduced a novel design for on-chip voltage sensors based on ROs: the JRO-based sensor. By enhancing sampling frequency and resolution of this family of sensors, we enabled their use for runtime voltage fluctuation measurements. To illustrate the performances provided by the JRO sensor, we adopted them to conduct power SCA attacks within an FPGA fabric. An adversary sensor shell was used to perform an FPGA-to-FPGA attack on a victim AES module located within a logically isolated shell. Thanks to the performance improvement brought by the JRO sensor we were able to perform the first CPA attack conducted using RO-based sensor within an FPGA. Successful results obtained in retrieving the secret key of the AES running at 50 MHz demonstrate the strong overall performances provided by the JRO sensor when used for side-channel purpose. To further evaluate the JRO, we compared it to different kinds of side-channel setups. A CPA attack was conducted using TDC-based sensors and an EM traditional side-channel setup. We showed that thanks to their proximity to the target, on-chip sensors may provide results similar to near field EM even with a much smaller sampling rate

and resolution. Regarding the integrated voltage sensors, the JRO-based sensor almost reaches the accuracy of TDC-based sensors and benefits from a lighter area overhead. Moreover, it has a better spatial coverage and an easier implementation as it relies on basic logic gates. Finally the JRO stands as an ideal alternative for monitoring fine-grained high-speed voltage fluctuations in FPGAs.

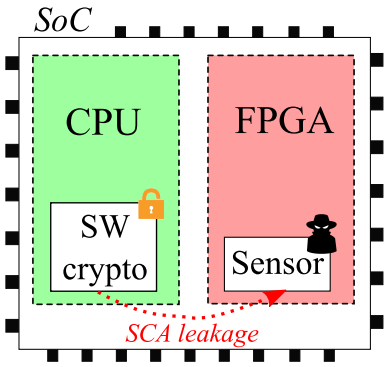


Figure 3.13 FPGA-to-CPU attack

3.4 FPGA-to-CPU - Remote Side-Channel Attacks on Heterogeneous SoC

The work described in this section was published in [50] with co-authors Jean-Max Dutertre, Yannick Teglia, Philippe Loubet Moundi and Francis Olivier.

The works conducted in the first section of this chapter contributed in exposing the multi-user FPGA threat. The FPGA-to-FPGA attacks conducted fulfilled the first step of this thesis roadmap that was introduced in chapter 1. In this second section, we aim at assessing the dangers that FPGA-based voltage sensing could represent in SoC systems. This second thesis step consists in building FPGA-to-CPU attacks on heterogeneous SoCs.

3.4.1 Introduction

This second work focuses on a specific application of FPGA-based SCA attacks. Using a heterogeneous SoC that consists in an FPGA and a CPU implemented on the same die, we carry on the work started by [148] which consisted in eavesdropping CPU computation using FPGA-based voltage sensors. The contributions of this section are detailed below:

- We provide an in-depth study and improvements of FPGA-based voltage sensors performances for side-channel purpose.
- We conduct the first FPGA-based SCA on symmetric crypto-algorithms running on the CPU core of a SoC: *Tiny AES* + *OpenSSL AES*.
- We evaluate and compare the performance of FPGA-based sensors with a traditional electromagnetic side-channel setup.

An iterative work was conducted from the reproduction of the actual state-of-the-art through the attack of a hardware AES module to the first successful attack of a software AES program. Subsection 3.4.2 describes the global side-channel setup. Subsection 3.4.3 and 3.4.4 are dedicated to the side-channel experiments conducted both on hardware and software AES implementations. Then, Subsection 3.4.5 provides EM SCA

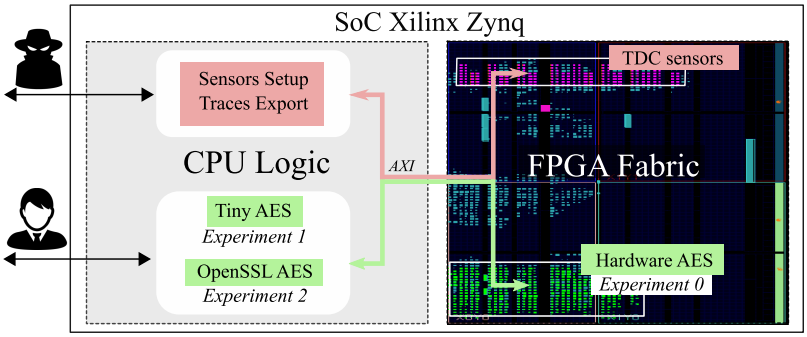


Figure 3.14 Xilinx Zynq experimental side-channel setup

results conducted on the same targets for comparison purpose and discuss the feasibility and countermeasures of FPGA-based SCA attacks. Finally, Subsection 3.4.6 concludes this work.

3.4.2 Presentation of the Side-Channel Setup

3.4.2.1 Side-Channel Targets

Previous works related to FPGA-to-CPU attacks on software targeted simple, self-written public key algorithms [148]. We propose to go a step further, by proving that freely-available (and actually deployed) symmetric crypto-algorithms are also vulnerable to FPGA-based SCA attacks. By taking advantage of the high accuracy and performances provided by TDC-based sensors, we aim to conduct CPA attacks against AES software implementations. *Please note that JRO-based sensors won't be used in this section as they were still in development at the time of these experimentations*.

The work conducted in this section targets one hardware AES and two software AES implementations. Each one of them implements distinct characteristics and enriches the global study. The first experiment targets a hardware AES module implemented within the FPGA fabric. The goal of this attack is the evaluation of the intrinsic device leakage and the optimization and calibration of the TDC-based sensors (note that this attack was already conducted by [122]). The second and third experiments are conducted on Tiny AES [78] and OpenSSL AES [106]. Experiments 2 and 3 represent the novelty of this work.

3.4.2.2 Xilinx Zynq Experimental Setup

The entire side-channel setup is based on a Xilinx Zynq 7000 heterogeneous SoC that implements both FPGA (Xilinx Artix-7) and CPU (ARM Cortex-A9) on the same die [141]. Figure 3.14 represents our experimental setup which is organized as follows: the Artix-7 FPGA fabric embeds 8 TDC-based sensors set to provide a sampling rate of 200 MS/s per sensor and all the logic required to store the acquired data (FIFOs). *The use of several TDCs increases the voltage fluctuation coverage area and the granularity of*

the overall side-channel setup. However, TDCs multiplication is limited by the voltage noise resulting from their own activity. 8 TDCs is the best trade-off found during our experiments.

The fabric also integrates a custom hardware AES-128 module implemented for experiment purposes. The dual-core Cortex-A9 CPU is cadenced at 667 MHz and runs a bare-metal C program that implements both Tiny and OpenSSL AES. From the attacker point of view, the side-channel traces are exported through UART for upcoming CPA computations. (*Note that in a practical scenario, CPA computation could be launched directly inside the target to reduce the amount of exported data*).

3.4.3 FPGA-based attack on Hardware AES

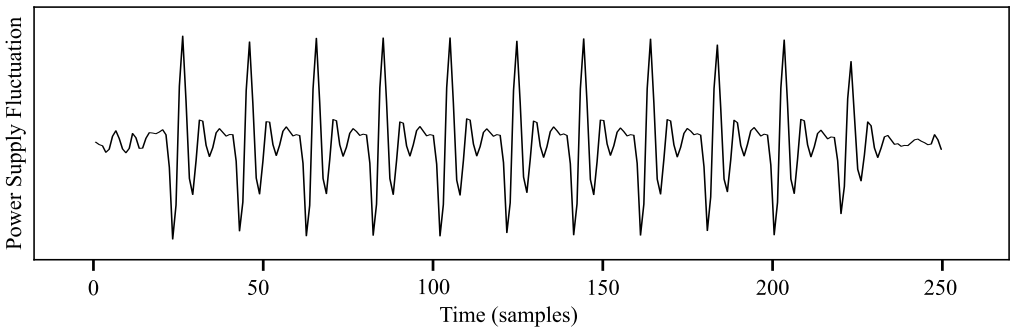


Figure 3.15 Averaged power supply fluctuation resulting from 100 hardware AES encryptions. AES frequency: 10 MHz - TDCs sampling rate: 200 MS/s.

We already conducted FPGA-to-FPGA attacks on hardware AES in the previous section using JRO-based sensors. In this work we reuse the AES module that was previously implemented however this time we study its power consumption using TDCs. Figure 3.15 illustrates the hardware AES power consumption measured using the 8 TDCs of our test setup (their output values are added and averaged).

Side-Channel Attack: As in Section 3.3, the CPA selection function was taken as the Hamming Distance between the 9th and 10th round register values and a 8-bit assumption was made on the last round key (*last round attack* [92]). We acquired 10,000 AES leakage traces using TDCs and conducted a CPA attack. The use of large set of traces progressively led to the extraction of the leakage out of the noise variance. And, after 4,483 traces on average, the right key was found. Despite the attack success, the SCA results are not optimal. Through the calibration of the TDCs, we believe that the number of traces required to retrieve the AES secret key can be significantly reduced. Two SCA setup optimizations are presented in the following paragraphs, their effect on CPA results is illustrated in table 3.2.

Placement optimization: The impact of the sensor distance from the target was already discussed in [122]. Here, we implemented it as a preliminary side-channel

TDC Calibration	Average number of Traces	Optimization Factor
No	4,483	/
Placement	3,440	1,30
Init + Placement	1,381	3,25

Table 3.2 TDC optimizations and their impact on the number of traces required to infer an AES key byte (averaged on its 16 bytes).

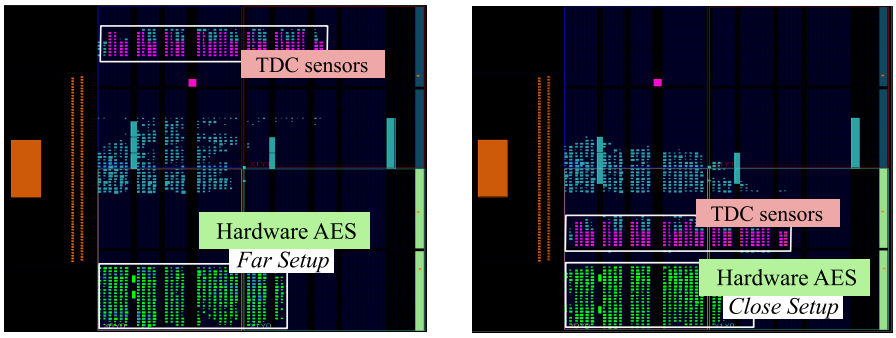


Figure 3.16 Logical distance between sensors and target algorithm.

optimization. Close and far setups were instantiated as depicted in Figure 3.16. In the far setup (used for the previous attack), the TDCs were 80 slices distant from the AES, while in the close setup, the logical distance between them was 6 slices.

As illustrated in table 3.2, by adopting the close setup, the number of traces required to perform the attack dropped from 4,483 to 3,440. It appears that the noise induced by the logic placed between the AES and the sensors alters the SCA signal quality. Moreover, the physical distance between the AES and the sensors leads to a natural attenuation of the SCA leakage amplitude. Therefore the distance between the sensors has a significant impact on the CPA results (here thirty percent less traces were required).

Init delay optimization: Init delay of the TDC represented in Figure 3.4 (and more specifically in Figure 3.21 in appendix) can be dynamically configured using coarse and fine tuning. In our experimentations, we programmed the dedicated logic (MUX) to modify the number of logic elements forming the path, and consequently the delay duration. The δclk edge propagation speed gets impacted by all the voltage fluctuation that occurs as it travels through the $init$ delay, yielding thereby an averaging effect. This effect naturally smooths the sampled values and thus acts as a high-frequency noise filter. However, depending on its duration, it can deteriorate the accuracy of the sensor.

Through the implementation of 4 delay paths having different lengths, we were looking forward to finding the best averaging trade-off for our device. The $init$ delay was increased of a half clk period per path. In practical terms, the $init$ path size (logic elements) was progressively increased until the propagation of the δclk edge filled a half of the delay-line with "1". When it was the case, a half clk period of delay had been added. This operation was repeated 1, 2 or 3 times depending on the chosen delay duration.

Experimentally, the SCA attack results were progressively enhanced with the $init$ delay path size, until it reached 1.5 times the clk period. Then, it finally decreased for the last setup. As highlighted in table 3.2, CPA results were significantly improved by the $init$ calibration, the number of traces required to infer the secret key dropped from 3,440 to 1,381 traces. Altogether, placement and $init$ delay calibration divided by 3,25 the number of traces required to infer the AES secret key. This optimization is substantial for the following attacks that require a significantly larger number of side-channel traces.

3.4.4 FPGA-based attack on Software AES

In this subsection, SCA attacks are conducted against freely available AES software implementations. The optimal setup for the attacks relies on 8 TDCs placed vertically along the left border of the fabric. According to the Zynq implemented design, this placement makes sense as it bring TDCs closer to the processing system (i.e CPU). While this work focuses on the FPGA-to-CPU SCA attack feasibility, the identification of the best TDC positions and shapes need to be further investigated in future works.

3.4.4.1 Experiment 1 : 8-bit Tiny AES

The first target adopted for CPU experiments was the Tiny AES implementation available on GitHub [78]. This small 8-bit data path AES computes each AES transformation sequentially and processes data from the less to the most significant byte. Our experiment focused on the AES-128 encryption, plaintexts were randomly generated and collected through UART. To make sure that the AES ran at the CPU max frequency (667 MHz), we measured the number of clock cycles elapsed during the encryption using ARM performance counters: around 26,000 clock cycles were required for a full encryption (39 μs). Figure 3.17 illustrates a full Tiny AES encryption acquired using TDCs at a sampling rate of 200 MS/s. The first 9 rounds of the AES can be easily distinguished thanks to the variation of power consumption between 8-bit AES transformations *SubBytes*, *ShiftRows* and *AddRoundKey* and the 32-bit *MixColumns* transformation. The last AES round differs

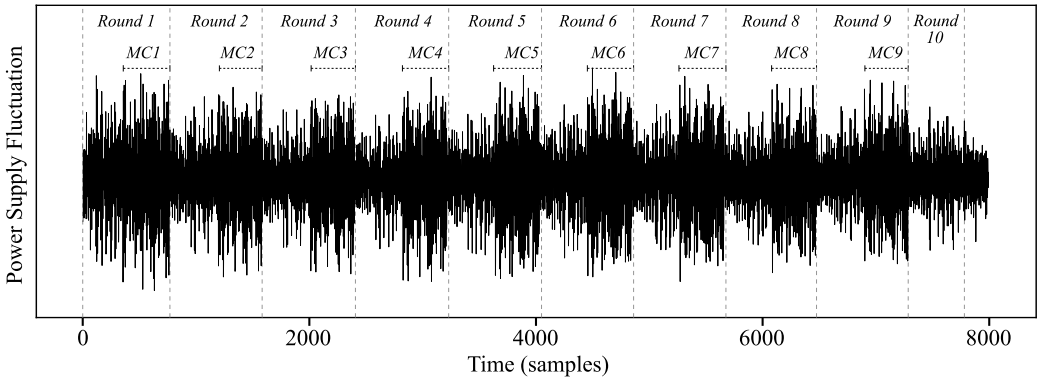


Figure 3.17 Averaged power supply fluctuation resulting from 100 Tiny AES encryptions. TDC sampling rate is 200 MS/s per sensor.

from the others as it doesn't use the *MixColumns* subfunction.

Side-Channel Attack: The side-channel leakage resulting from 8-bit AES computations has a low impact on the overall chip voltage fluctuations. The encryption measurement only covers 5 TDC quantization levels among the 32 possible and is thus more vulnerable to the low-frequency noise induced by the surrounding peripherals (e.g., voltage regulator module 500 KHz) and physical effects (e.g., temperature variations). To enhance the signal-to-noise ratio and reduce the number of traces required for the attack, we needed to apply a high-pass filtering on each side-channel trace. After preliminary filtering, the CPA could be conducted. The selection function chosen for the CPA was the standard Hamming Weight model of the first round *SubBytes* output: $HW[Sbox[k \oplus m]]$. The attack was a success, an average of 111,000 traces were required to infer a secret AES key byte. Despite a significant increase of the number of traces required for the attack (compared to the hardware AES attack), TDCs were still accurate enough to perform CPA against software algorithms running on our target. The side-channel performance deterioration can be explained by the greater logical distance between the FPGA-based sensors and the CPU logic and the lower sensitivity of the TDC-based sensors.

3.4.4.2 Experiment 2 : 32-bit OpenSSL AES

The OpenSSL library [106] implements a wide range of cryptographic algorithms massively used for secure channels over computer networks. In this work, we focused on the OpenSSL AES-128 that implements a 32-bit tabulated version of the AES [30]. This variant merges the *MixColumns* and *SubBytes* transformations into 4 look-ups tables known as T-tables (256 x 32-bit). The round transformation of each input byte is directly loaded from the T-tables and thus speeds up the computations. OpenSSL cadenced at 667 MHz encrypts 128-bit of data in 2,9 μs - 13.5 faster than the Tiny AES. Figure 3.18 illustrates the power consumption induced by 1,000 OpenSSL AES encryptions. In this experimentation, the OpenSSL encryptions were placed in between two empty for loops to facilitate their visualization. This time the 10 AES rounds cannot be easily spotted because of the

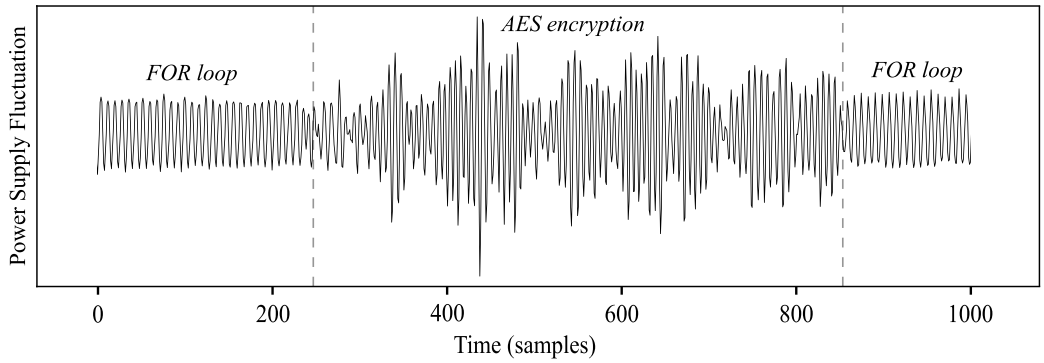


Figure 3.18 Averaged power supply fluctuation resulting from 100 OpenSSL AES encryptions. TDC sampling rate is 200 MS/s per sensor.

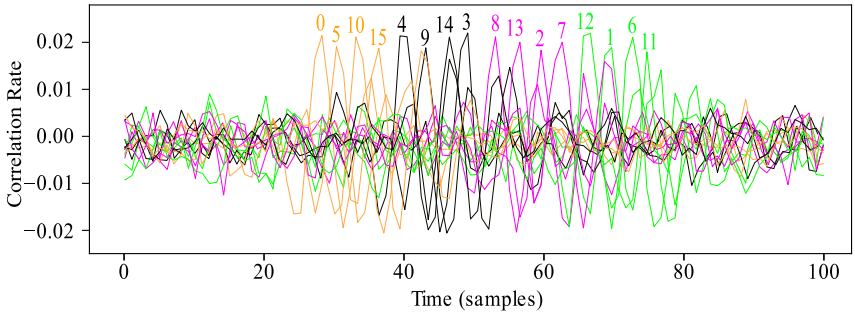


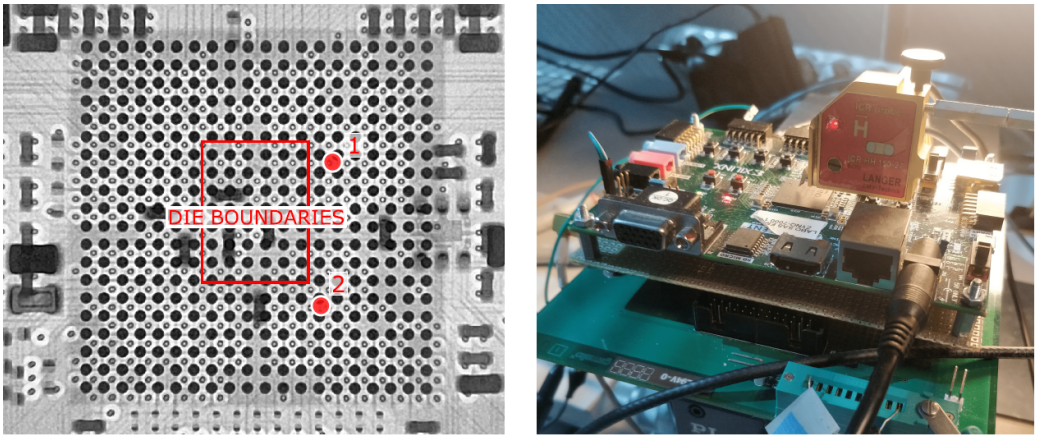
Figure 3.19 Correlation rate over the time obtained for the good guess of each OpenSSL key byte. The 32-bit implementation can be recognized by observing the byte order. (Each color represent a 32-bit word)

T-table OpenSSL implementation that only consists in 32-bit operations.

OPENSSL1: 8-bit selection function. The first round model $HW[Sbox[k \oplus m]]$ previously used for the Tiny AES attack works fairly well even against OpenSSL tabulated AES. According to the definition of the 4 T-tables described in [30], each table output consists in a 32-bit word $T[k_i \oplus m_i]$ in which the natural *Sbox* value relative to the input byte $k_i \oplus m_i$ was multiplied by the *MixColumns* coefficients. For each table the natural 8-bit value of the *Sbox* appears twice in the word because two of the *MixColumns* coefficients equal one. Therefore, 16-bit of the 32-bit output word will leak according to the *Sbox* model. Using such a selection function, 130,000 traces were necessary for the attack to succeed.

OPENSSL2: 32-bit selection function. The first round model can be used to perform reverse engineering. Contrary to classic 8-bit AES which computes each AES byte from the less significant byte to the most significant, tabulated 32-bit AES computes each key byte according to the *ShiftRow* order. This order can be perceived in temporal CPA results. Figure 3.19 illustrates the timing correlation obtained for each right guess of the 16 AES key bytes. The byte order follows the *ShiftRow* order and betrays the presence of a tabulated AES. Thanks to this information, an attacker can slightly improve the CPA model by making a full 32-bit prediction. Instead of targeting the Hamming Weight of the *Sbox* output, the attacker adds the T-table in his selection function: $HW[T_t(k \oplus m)]$. The expected benefit is a slightly better correlation and a quicker hypotheses distinguishing. Experimentally, the average number of traces required to perform the attack dropped to 87,000 which is 1.5 time less than the *Sbox* model.

This subsection experimentally demonstrated that FPGA-based sensors are suitable for SCA attacks against software symmetrical algorithms. According to the selection function adopted for the experiments the number of traces required to infer the secret key fluctuated from 87,000 to 127,000. No significant distinction exists between the 8-bit Tiny AES and 32-bit OpenSSL AES attacks as they both leaks accordingly to the *Sbox* model. The attack of side-channel resistant crypto-algorithms could be considered in future works to further evaluate the potential and limitations of FPGA-based sensors.



a) BGA Zynq XRAY view *b) EM side-channel Setup*

Figure 3.20 On the left, an XRAY picture of the Zynq BGA package, the die is contained within the rectangle. On the right, the side-channel setup based on a langer EM probe.

3.4.5 EM Results & Discussion

To evaluate the pertinence of SCA that can be performed remotely using TDC-based sensors, we challenged these results regarding classical local SCA attacks. As done for JROs in Section 3.3, this subsection presents a performance comparison with a traditional EM side-channel setup and discusses about FPGA-based attacks feasibility and associated countermeasures.

3.4.5.1 Electromagnetic Side-Channel Attack

Figure 3.20.b illustrates the adopted EM setup that consisted in a Langer near field microprobe connected to an oscilloscope with a 5 GS/s sampling rate and a 12-bit resolution. A XRAY picture of the ZYNQ7000 depicted in Figure 3.20.a was taken to check the die structure. Two hot-spots are represented, the first one offered the best contrast and visualization of the hardware AES side-channel leakage, while the second one gave the best results for software AES algorithms. The EM leakage of the first round of each attacked AES was leveraged to trigger the oscilloscope. The captured samples were then extracted and used to perform a CEMA.

CEMA was conducted against each AES implementation studied in this section. Table 3.3 gathers all the results obtained with both TDCs and EM setups. According to table 3.3, the hardware AES and OpenSSL AES attacks based on TDCs required roughly as

Setup	HAES	Tiny AES	OpenSSL 1	OpenSSL 2
EM	**1,021**	**52,438**	**106,225**	88,412
TDC	1,381	111,758	127,558	**87,422**

Table 3.3 Averaged number of traces required to retrieve a key byte on various AES implementations for EM and TDC side-channel setups

many side-channel traces than EM. This means that with only 32 quantization levels and a 200 MS/s sampling rate, TDCs provided similar results to a high performance oscilloscope. Naturally and as concluded for JRO in Section 3.3.5, this must be interpreted with caution as TDCs were previously calibrated and optimized for this specific device and attack scenario. Moreover, a significant difference between EM and TDCs still appears when it comes to the Tiny AES attack. As mentioned before, this probably has to do with the sensitivity limitation of the TDCs.

This experiment demonstrated that through the calibration and optimization of our sensors, we are able to obtain similar results to traditional side-channel setups. Therefore, this further proves the extents of the SbSCA attack threat in FPGAs.

3.4.5.2 Attack feasibility

The feasibility of FPGA-based attacks on a practical scenario substantially relies on the security level provided by the target. Three major requirements need to be met to make it possible:

1) Information Leakage Medium: The SCA attacks conducted in this section required the implementation of voltage sensors within a victim FPGA fabric. This can be done in cloud datacenters through the rental of reconfigurable logic, by the insertion of malicious trojan within untrusted FPGA IPs or through the direct reconfiguration of unsecured FPGA chips.

2) Data knowledge: SCA attacks conducted against secret key algorithms such as AES require the knowledge of victim plaintexts or ciphertexts. Depending on the use case, accessing this information can be challenging for the attacker especially because each trace acquired using TDCs must match with the exact plain/cipher text used for the encryption. The feasibility is related to the opportunity for the attacker to trigger victim encryption and to retrieve plain or cipher texts.

3) Synchronisation: Victim side-channel leakage needs to be dynamically detected by the sensor logic to facilitate the attack. A trigger mechanism can be implemented within the TDC to start the data storage when a large voltage undershoot occurs. However, this trigger mechanism cannot be fully reliable and sometimes gets disturbed by surrounding noise induced by temperature variations or peripherals computations. Depending on the overall noise level, the attack complexity can soar. To facilitate the attack, a local clone of the targeted device can be used to adjust and calibrate the side-channel setup towards the actual remote exploit.

3.4.5.3 Countermeasures

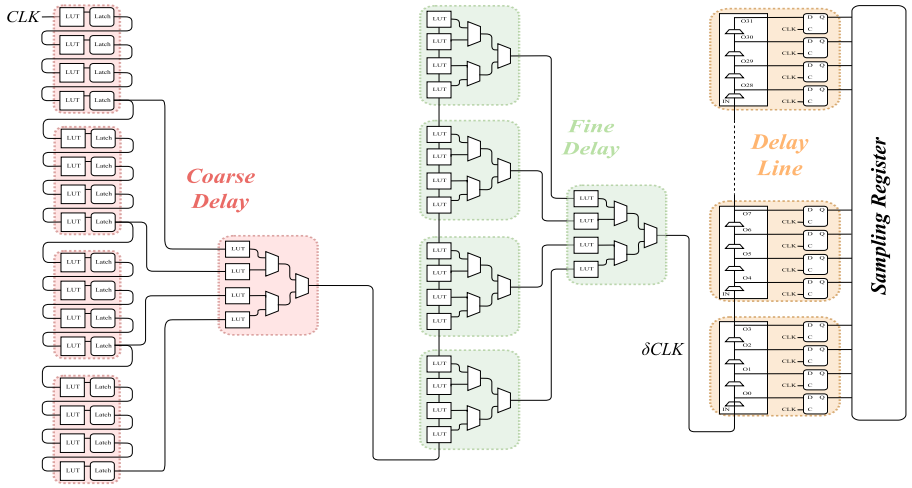
The threat behind FPGA-based hardware attack has already been taken into account by cloud providers who assure that, for the sake of security and integrity, their FPGA resources are not shared between multiple users. However, although this protection mitigates intra-FPGA attacks, FPGA sensors can still eavesdrop computations that occur in

other chips connected to the same power supply even in presence of decoupling capacitors [121]. To mitigate the threat once for all, an independent power supply should then be provided for each FPGA chip. Protecting SoCs that implement both FPGA and CPU within the same die should be more complex. The dissociation of the power sources would require the creation of two independent power distribution networks and thus increase the overall design cost. Designers should be aware of the side-channel threat and should consider it even when the device is not physically accessible by the attacker. An efficient way to prevent a crypto-algorithm from being remotely attacked is the usage of the usual side-channel countermeasures as for instance shuffling, masking, random delays or jitter [147, 149].

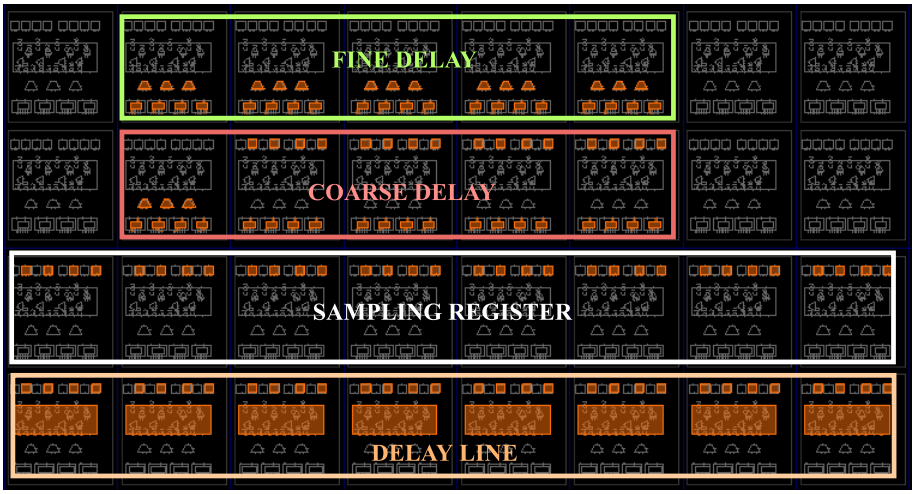
3.4.6 Conclusion

With the massive adoption of FPGA hardware acceleration in connected systems such as SoC and cloud data centers, the eventuality of remote FPGA-based hardware attacks become more and more realistic. In this section, we demonstrated that FPGA logic instantiated within a complex SoC can be leveraged to monitor voltage fluctuations of the surrounding logic blocks and in particular that of a CPU. We conducted three experiments from the SCA attack of a hardware AES instantiated within the FPGA logic to the attack of two software AES running on the CPU core (FPGA-to-CPU attack). The first experiment was carried out on the hardware AES module. It allowed us to calibrate several parameters to improve side-channel results (init delay, position, filtering, etc.). Then we performed the first FPGA-based SCA attacks on software AES (Tiny AES and OpenSSL AES). The side-channel leakage induced by the CPU core being much weaker, the attack required a substantial increase of the number of side-channel to infer the encryption key. To evaluate the performances of our sensors, we conducted the same attack using a EM traditional setup and obtained comparable results to those achieved with TDC-based sensors. This attests the extend of the threat that unsecured FPGA SoC constitute. Finally, care must be taken when designing SoC to ensure that hardware resources cannot be maliciously used as hardware attack means.

3.4.7 Appendix



a) Logic schematic of the TDC-based sensor



b) FPGA Implemented design of 1 TDC instance

Figure 3.21 Logic schematic and implemented design of one TDC-based sensor instance. Each dotted rectangle in the logic schematic represents 1 slice (26 in total). The delay-line provides 32 quantization levels and a sampling rate of 200 MS/s per sensor.

Figure 3.22 SCAbox Logo

3.5 SCAbox - A Framework for Evaluating the FPGA-based SCA Threat

The work reported in the FPGA-to-FPGA and FPGA-to-CPU attack sections led to two major contributions. The creation of a novel RO-based sensor (JRO) optimized for FPGA-based SCA and the implementation of statistical attacks on software symmetric crypto-algorithms. Before entering the third step of this thesis dedicated to the implementation of SbHA without the use of an FPGA, we present a framework dedicated to facilitate the implementation of FPGA-based SCA attacks for research purposes.

This section introduces SCAbox: an FPGA-based SCA framework dedicated to research and educational purposes. SCAbox bases itself on recent academic works that use FPGA-based sensors as side-channel vectors to eavesdrop the power activity of hardware and software cryptographic algorithms. The reproduction of this new side-channel method is facilitated by SCAbox as it gathers sensors implementations, cryptographic targets and an architecture to collect the side-channel leakage. The entire framework is open-source and maintained in a public GitHub repository. It has a modular architecture that makes it possible to carry out various side-channel experiments using different sensors and crypto-algorithms.

In this section we focus on describing the architecture of SCAbox and demonstrate a typical use case based on a TDC converter sensor and an AES hardware module. To ease the user experience, this work also links to a website with demonstrations and extended tutorials: https://emse-sas-lab.github.io/SCAbox/.

3.5.1 Introduction

SCAbox has been designed for the Xilinx Zynq SoC family that embed both processors and FPGAs within the same silicon die. The Zynq architecture makes it possible to build digital voltage sensors using the FPGA logic and use them to perform SCA attacks on assets located either in the FPGA (hardware IPs) or in the processor (software applica-

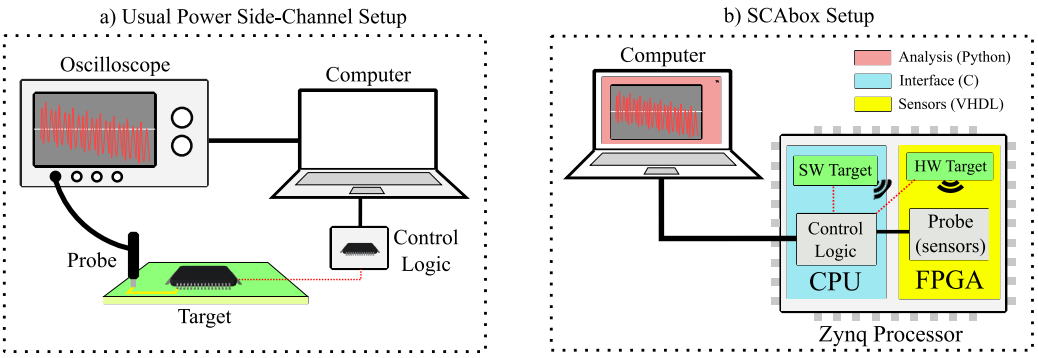


Figure 3.23 Reducing SCA setup complexity: comparison between usual and SCAbox configurations

tions). SCAbox aims at popularizing hardware security by enabling SCA experiments at low-cost using SbHA. By using the SCAbox framework you will be able to:

- Familiarize with SCA and co-design development.
- Reproduce attacks conducted in recent academic papers.
- Build your own FPGA-based side-channel sensors.
- Characterize the SCA leakage of your hardware and software algorithm implementations.

The SCAbox framework is described in Subsection 3.5.2 and a proof on concept is demonstrated in Subsection 3.5.3. Subsection 3.5.4 concludes this section.

3.5.1.1 Comparison with Usual Side-Channel Attacks

Figure 3.23.a illustrates a typical side-channel setup that uses an oscilloscope and a probe to collect the target power activity. This setup requires control logic to communicate with the target and a computer to collect the oscilloscope traces and perform the analysis. With this kind of setup, the SCA attack can be fully automatized but its construction is complex and makes it rather difficult to take in hand for beginners. Light platforms such as ChipWhisperer [103] reduce the complexity by merging the oscilloscope, the probe and the control logic into a single device reducing then the element number from five to three.

As depicted in Figure 3.23.b, FPGA-based SCA further reduces this number by embedding all elements except the computer within a Zynq processor. Although this limits the hardware skills required to conduct SCA, it also supposes that the user is familiar with programming in VHDL and C languages. To limit this assumption, SCAbox aims at offering an easy-to-use platform to help researchers in creating various FPGA-based SCA applications.

3.5.2 Framework Architecture

3.5.2.1 Overview

The SCAbox structure is illustrated in Figure 3.23.b and can be divided into three main blocks. The first one in yellow is implemented within the Zynq FPGA. It contains the sensors, the storage mechanism and the victim hardware algorithm. The second one in blue is located in the Zynq CPU. It implements a C program dedicated to communicate with the FPGA and to extract the SCA data collected through a serial port. The program also implements optional victim software algorithms that can be subjected to SCA attacks. The last block in red is launched from the acquisition computer. It consists in a Python-based application capable of connecting through serial communication to the Zynq board, collecting SCA acquisitions and conducting SCA attacks on the targeted algorithms.

The first release of SCAbox contains the IP cores and applications depicted in the table below. The user can choose between two sensor IPs (TDC or RO-based sensors) to monitor the activity of various targets. Three hardware targets (AES, Present and Klein block ciphers) are provided along with three different software implementations of the AES algorithm. The internal SCAbox software and hardware architecture is described in the following paragraphs.

Setup	Hardware IP cores	Software Programs
Sensors	TDC [54] RO [50]	/
Targets	AES PRESENT KLEIN	Tiny AES Dhuertas AES OpenSSL AES

3.5.2.2 Hardware Architecture (FPGA)

As depicted in Figure 3.24 (left), the hardware architecture of SCAbox uses four major IP cores.

- The *Zynq IP* is used to interface the FPGA logic with the processor. It maps the IP core memory addresses to enable communication with the software program.

- The *Storage IP* is controlled using software drivers from the processor. According to a `start` signal launched from software, it triggers the storage of sensor data into a FIFO. When the acquisition is running, the FIFO collects a new sensor value at every clock cycle. Once the acquisition stops, the data is extracted from the FIFO to the processor in order to proceed with parsing and serial export.

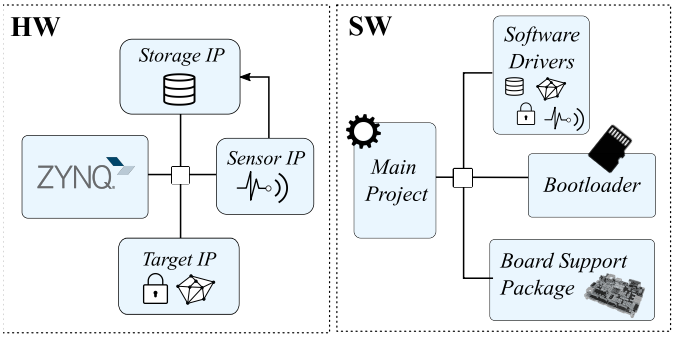


Figure 3.24 SCAbox hardware (VHDL) and software (C) architecture

- The *Sensor IP* is designed to collect an image of the actual on-chip voltage. It can be made out of different logic blocks such as ROs or TDCs. Sensors are programmed through software drivers from the processor and have their output connected to the storage IP.

- The *Target IP* is deployed to be analyzed through SCA. The initial release of SCAbox comes with several cryptographic IPs that can be used to evaluate custom sensors and to perform SCA attacks. Any SCAbox IP can be replaced. It is also possible to implement multiple targets within the same block design. However, every design change will imply software architecture modifications.

3.5.2.3 Software Architecture (Processor)

Figure 3.24 (right) illustrates the Zynq processor program organisation. The bare-metal *Main Project* uses IP cores *Software Drivers* to control the underlying hardware and provides an interface with the outside world through a serial communication. Using a serial prompt the user can navigate through the available Application Programming Interfaces (APIs) that enable communication with the FPGA IP cores. The available commands are sensor and target specifics. They allow the user to test and collect the sensor outputs, to run the implemented algorithms and to perform SCA acquisitions. Additionally to these features, the main program embeds software algorithms that can be used as targets for the FPGA-based sensors. The installation of the Xilinx Vitis software development kit (SDK) is required to modify the commands and algorithms implemented. Tutorials on how to adapt the software application according to the underlying hardware are described in the SCAbox website.

3.5.2.4 SCA Automation Tool (Computer)

To facilitate the data acquisition and visualisation, SCAbox comes with a simple application built with Python. This application connects directly to the Zynq board through a serial communication and can exchange data and commands with the device. The user interface facilitates serial command construction and automatically displays the collected

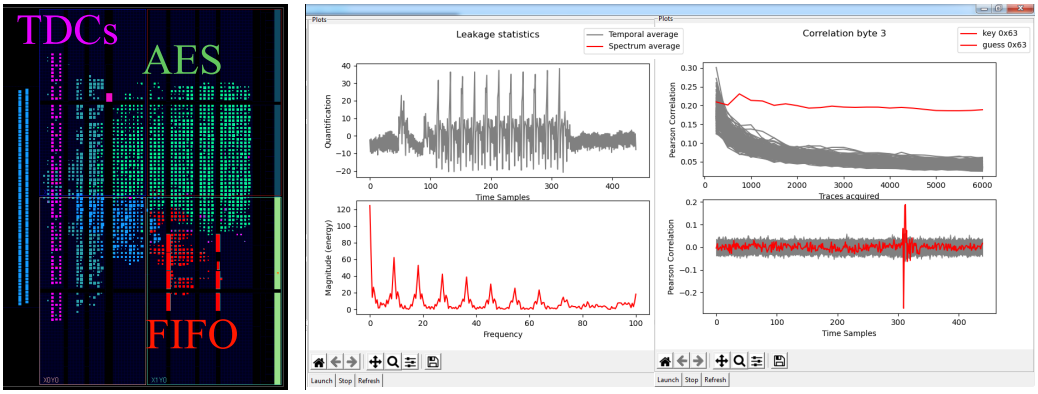


Figure 3.25 Use case FPGA implemented view and CPA results obtained using the SCAbox automation tool.

SCA traces. Moreover, the initial application release also features the well known CPA attack on AES [18]. This means that the user can conduct a full AES key retrieval using FPGA-based sensor acquisitions.

The SCA automation tool is not required to build the SCAbox project. It has been designed for demonstration means and can be replaced by any other SCA tool able to parse the data received from the serial port.

3.5.3 SCAbox User experience

The SCAbox framework aims at being user friendly. Even if co-design can be facilitated with predefined IPs, it still relies on complex development environments and on the knowledge of multiple programming languages. For this reason, we propose several tutorials with gradual difficulty levels from the getting started demonstration that avoid any programming to the most advanced that consists in building custom target and sensor IP cores. In the following paragraphs we depict an SCAbox use case that focuses on reproducing the works presented in the previous sections on FPGA-to-FPGA and FPGA-to-CPU attacks.

3.5.3.1 Use case: SCA Sensor & Setup

This demonstration uses TDCs as sensors and a hardware AES algorithm as a target (as described in Section 3.4). The idea is first to measure the AES power consumption and then conduct a SCA attack to infer its secret key. The left part of Figure 3.25 represents the implementation view for this SCA setup. In purple, eight TDCs have been placed. The remaining logic implements the Zynq processing system, the AES running at 10 MHz and the FIFO controller. Each TDC provides 32 quantization levels (n) and is cadenced at 200 MHz. *A boot image containing the SCA setup bitstream and the application dedicated to this use case can be directly downloaded from the GitHub repository and loaded into a micro SD card.*

3.5.3.2 Use case: SCA Acquisition & Attack

The automation Python tool described in Subsection 3.5.2.4 is employed to collect the TDC acquisitions and conduct a CPA attack. A configuration window helps the user in building serial commands to communicate with the Zynq platform. The user can select the targeted algorithm, the number of acquisitions, the sample range, etc. The right part of Figure 3.25 represents the AES acquisition results captured and displayed using the automation tool. The average power consumption collected appears progressively on the plot window located in the top left hand corner. The AES power consumption can be easily recognized thanks to its ten characteristic rounds that create high power consumption spikes. The magnitude spectrum plotted in the bottom left hand corner indicates the leakage amplitude at each frequency. Here, we can observe multiple of 6.66 MHz harmonics that are probably linked to the processor activity (CPU frequency: 666 MHz).

As in Section 3.4.3, the CPA attack is conducted on the last AES round. Two plots illustrate the CPA attack results for a specific key byte (here byte 3). On the top right hand corner, we can see the correct key candidate (in red) emerging from the other candidates (in gray) after less than 500 acquisitions. On the bottom right the temporal representation indicates that the key leaks around the time sample 300. Finally, the entire key can be retrieved with less than five thousand AES encryptions. This simple SCAbox demonstration successfully reproduces the results obtained in Section 3.4 and illustrates the capabilities of FPGA-based SCA attacks.

3.5.3.3 Discussion

This paragraph addresses the SCAbox strengths and limitations and more importantly its purposes.

What does SCAbox offer that is new?

- A software-based platform to perform power SCA attacks. No need for strong hardware skills.
- A direct view of the internal chip voltage fluctuations. Not filtered, that is, not damaged.
- The possibility to evaluate various algorithms on equal footing for fair comparison between implementations.

What's SCAbox suitable for?

- Reproduce existing FPGA-based SCA attacks.
- Familiarize with SCA and improve co-design skills.
- Evaluate the SCA leakage of VHDL hardware designs.
- Evaluate the SCA leakage of C software libraries.

What's SCAbox not suitable for?

- Conduct high sampling rate SCA. The actual maximal FPGA-based sensors frequency is limited to 250 MS/s.

- Evaluate the SCA leakage of an external device. SCAbox only operates on internal algorithms.

3.5.4 Conclusion

SCAbox aims at facilitating FPGA-based SCA by providing an open-source framework that gathers sensor implementations, cryptographic targets and an architecture to collect SCA leakage. This section introduced the SCAbox framework architecture and demonstrated a simple use-case. The SCAbox framework is dedicated to hardware security enthusiasts which want to familiarize with SCA attacks and to researchers that aim to evaluate the leakage of hardware and software designs. More information on SCAbox can be found on GitHub: https://emse-sas-lab.github.io/SCAbox/.

3.6 Conclusion on FPGA-based Power Analysis

In this chapter, we described three works dedicated to the study of FPGA-based power SCA attacks. Each work brought a different kind of novelty either on the sensor designs, the attack scenarios or the method for helping researchers in reproducing these attacks. Beyond the results themselves, this study helped us in demystifying on-chip power analysis, understanding its limitations and making a step toward SoC attacks.

3.6.1 Results Reminder

In this subsection we establish a quick reminder on the contributions brought by the 3 works described in this chapter.

3.6.1.1 Main Contributions

These three works conducted on FPGA led to:

- The creation of an optimized RO-based sensor design named JRO. This is the first RO-based sensor suitable for statistical FPGA-based power analysis.
- The first statistical SCA attacks ever conducted from an FPGA to a CPU.
- The implementation of an open-source framework available on GitHub to facilitate the reproduction of FPGA-based power analysis works.

3.6.1.2 Some Numbers

Here we lay out some numbers to remind FPGA-based sensor performances.

Sensor Characteristics:

- TDC and JRO sensors can operate at **250 MHz** frequency in Artix-7 devices.
- A single JRO sensor consists in **2** FPGA slices.
- A single TDC sensor consists in **26** FPGA slices.
- A single TDC sensor has **32** quantization levels.
- A single JRO sensor has **3** quantization levels.

Attack Results:

- **64 JROs** can retrieve a hardware AES key in **7,685** SCA traces (@50MHz).
- **8 TDCs** can retrieve a hardware AES key in **1,381** SCA traces (@50MHz).
- **8 TDCs** can retrieve a software TinyAES key in **110,000** SCA traces (@666MHz).
- **8 TDCs** can retrieve a software SSL AES key in **87,000** SCA traces (@666MHz).

These results are **equivalent to those obtained using a local near field electromagnetic setup**.

3.6.2 *SbHA Knowledge*: Demystifying On-Chip Power SCA

The major SbHA properties and observations that were identified through the works conducted during this thesis are described within each chapter conclusions and denoted with the mention "SbHA knowledge".

The works conducted in this chapter aimed at evaluating SbSCA feasibility. Despite the challenges faced to mount FPGA-based power analysis attacks, we were able to confirm their potential threat for connected devices. Here we look back on three major SbSCA properties learned from our works conducted on FPGAs:

3.6.2.1 On-Chip Power SCA can be conducted using digital logic

An interesting point linked to this FPGA study is that the sensors built for conducting power SCA attacks were based on digital logic. This demonstrates that not only mixed-signal chips embedding analog entities are susceptible to this type of attacks. Each device containing a digital or an analog circuit that has a relationship with on-chip voltage is a potential victim. The conversion from voltage to delay is particularly interesting since modern SoCs embed delay elements for various purposes. Chapter 4 describes how some of these elements can be used as voltage sensing vectors in SoCs.

3.6.2.2 On-Chip Power SCA can be conducted with limited sensors

Since the apparition of SCA attacks, the oscilloscope and probe capabilities have been increasing along with the speed of Integrated Circuits (ICs). Gigahertz cadenced IC leakage is now often captured using gigahertz sampling frequency oscilloscopes in order to capture every clock cycle of the target. For this reason it is sometimes assumed that an SCA attack could only work if the sensing sampling rate was higher than the IC clock frequency (i.e. Nyquist-Shannon sampling theorem[5]).

The works described in Subsection 3.3.4.3 demonstrated that this statement is not necessarily true. Indeed, the leakage induced by transistor's switching activity lasts over several clock periods and can be acquired at a slower sampling rate. This enables SCA attacks against high-speed targets such as in the FPGA-to-CPU scenario where the CPU is cadenced @666MHz and the FPGA-based sensor @200MHz. In Chapter 4 we show that SCA attacks remain feasible with even more limited sensing capabilities.

3.6.2.3 On-Chip Power SCA can be conducted across the SoC boundaries

As its name suggests, a SoC consists in several hardware components implemented on the same silicon die. Through our experimentations, we were able to observe that the side-channel leakage induced by a component leaked throughout the entirety of the SoC

[5] https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

die. The closer the sensor, the higher the captured leakage intensity. However, even in the FPGA-to-CPU attack scenario, with the addition of CPU noise and physical distance, we were still able to successfully retrieve the AES leakage. These results are promising to mount attack scenarios on SoC devices and will be confirmed in Chapter 4.

3.6.3 *SbHA Knowledge*: Strengths and Challenges of On-Chip Power SCA

Through this work on FPGAs we recorded several on-chip power SCA strengths that can be leveraged to facilitate the attack but also challenges that must be overcome to efficiently build SbSCA attacks. They are related to the resources used for collecting the SCA leakage and the differences with local attacks in terms of feasibility.

3.6.3.1 Challenge 1: Trace Synchronization

As demonstrated in 3.3.5 and 3.4.5, FPGA-based sensors can compete with traditional EM SCA setup. However, they are not necessarily as precise as Analog-to-Digital Converters (ADCs) in oscilloscopes. Thus, the captured signal can be difficult to interpret since the sensor granularity is non-optimal. For instance, a single TinyAES trace in the FPGA-to-CPU attack scenario undetectable through the noise and thus difficult to resynchronize. To conduct statistical attacks in our experiments, we synchronized the AES execution with the sensor capture by launching them simultaneously. This method allowed us to get rid of the synchronization issues but could be a limiting factor if the attacker doesn't control the victim execution. Several other methods could be used to detect victim execution such as sensor trigger detection if the victim leakage is consequent (used in FPGA-to-FPGA scenarios) or cache side-channel for detecting victim execution (FPGA-to-CPU scenario).

3.6.3.2 Challenge 2: Data Storage

Statistical SCA attacks can be memory-consuming when a large number of leakage traces should be captured. In the FPGA-to-CPU scenario, we captured 200 TinyAES samples multiplied by 200,000 traces. With a sample size of 8-bit this represents 320MB of data stored in DRAM. For longer traces it would become impossible to collect and store all the traces in memory. To minimize this storage issue, we used accumulators and instead of storing 200 hundred thousands traces we only stored 256 average traces for each plaintext class. In addition to being super efficient, this CPA optimization is particularly useful in such memory limited systems. However, it is only usable if the collected traces are perfectly aligned and could be harder to conduct in desynchronized scenarios. For this reason, it is even more mandatory to find a reliable synchronization medium before launching a statistical SbSCA.

3.6.3.3 Challenge 3: Data Export

Considering the attack itself, the adversary can choose between extracting and exporting the traces in order to perform the SCA on a remote server or conduct the analysis directly within the target. In our experiments, we decided to extract the data through UART but they could also be extracted using a remote SSH session.

In a real-world attack scenario, an adversary could choose to export every collected traces but this could be easily detected by the target and slow down the overall attack. While the FPGA works presented necessitated the extraction of all the curves, we now prefer the more elegant integrated SCA version. With it, the attacker computes the CPA directly inside the target and only has to extract the key. Therefore, he reduces the data transfer from gigabits to some bits. If the analysis cannot be conducted internally or requires more computing resources, the first solution remains feasible.

3.6.3.4 Strength 1: Sensor Combination

Sensing capabilities in SbSCA can be improved by increasing the number of sensors used. If several sensing units are available, their contributions can be summed to improve the granularity of the overall leakage capture. We especially leveraged this effect when using JRO sensors by placing 64 of them in the fabric. Each sensor added brought a different kind of information that enhanced the overall signal reconstruction. This addition of different sensing sources will be also used in SoCs in Chapter 4.

In SbSCA, it can be interesting to use different sensing sources and leverage them for distinct purposes. For instance, a temperature sensor would be too limited in frequency to conduct a statistical SCA but it could be used as a way to detect the beginning of an encryption and then trigger a voltage sensing mechanism. The adversary should compose its experimental setup with the existing resources and take advantage of the distinct types of information brought by each SoC entity.

3.6.3.5 Strength 2: Unlimited Attack Time

A major difference between the SbSCA presented and traditional local SCA is that the attacker is not necessarily limited by time. During the attack, the victim remains in possession of its resources and therefore, the attack can last for days or months. While a local attacker should be fast enough to collect target secrets before the victim finds out that it disappeared, a remote attacker can launch attacks for long periods without being detected. Therefore, even if an SbSCA requires more time for prototyping, synchronizing and collecting data, this is not necessarily an issue since the attacker is not detected. This change of paradigm should be taken into account to evaluate and rate the SbHA attacks in general.

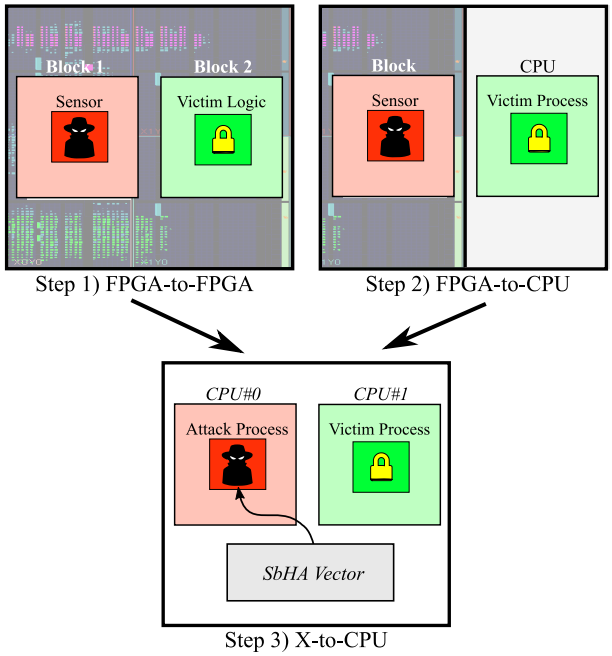


Figure 3.26 From FPGA to SoC SbHA exploits

3.6.4 A Step Toward SoC Attacks

As depicted in Figure 3.26, the conclusion drawn from these FPGA-based power SCA exploits paved the way to the apparition of SoC attacks. By introducing this new kind of attack, researchers broke the assumption that hardware attacks could be only conducted locally. Just a year after the first FPGA-based power SCA exploit, various novel SbSCA attack were disclosed. The next chapter will be dedicated to one of them, SideLine, a new kind of SbSCA that can be launched on any processor without the need of an FPGA.

Chapter 4. Software-based Power Analysis Attacks on Complex SoCs

Abstract

In this chapter, we describe *SideLine*, a novel side-channel vector based on generic delay-line components widely implemented in high-end System-on-Chips (SoCs). We demonstrate that these entities can be used to perform remote power SCA attacks on high-end devices and we detail several attack scenarios in which an adversary process located in one processor core aims at eavesdropping the activity of a victim process located in another core. In contrary to the FPGA-based attacks conducted in the previous chapter, *SideLine* does not require any reconfigurable logic. It is built on unsuspected sensors concealed in the hardware.

Chapter Contents

4 Software-based Power Analysis Attacks on Complex SoCs 89

4.1 Chapter Introduction 90

4.2 Technical Background 95

4.3 SideLine: Delay-Line-based power SCA on complex SoCs 99

4.4 Additional Results 115

4.5 Conclusion on Delay-Line-based Power Analysis 120

4.6 Appendix 125

4.1 Chapter Introduction

The works conducted in Chapter 3 confirmed the extent of the Software-based Side-Channel Analysis (SbSCA) threat but were limited to Field-Programmable Gate Arrays (FPGAs) and heterogeneous devices embedding FPGAs. In this chapter, we aim at generalizing SbSCA attacks to any SoC devices. Because most SoCs don't provide reconfigurable logic to implement custom hardware designs, it is necessary to identify components suitable for voltage sensing that are already implemented within the targeted devices.

In this introduction, we detail the modus operandi that led to the discovery of SoC delay-line voltage sensing capabilities. The strategy adopted to identify internal sensors included several steps. First, a set of targeted devices was chosen. Then, their technical documentations were studied to spot voltage sensing components. Finally, the selected sensors were compared, tested and used to mount SbSCA attacks. The set of target chosen for these experiments was composed of complex devices made out of multi-core processors and powerful enough to run a complex OS such as Linux or Android. The challenge was of consequence as at the time we began these researches, no power SbSCA had ever been conducted against complex SoCs.

4.1.1 Identifying SbSCA Vectors

The method employed to identify and select the best SbSCA vector among various potential targets was built on 4 main steps:

1. Identify **potential sensing components** in the target's data-sheet.
2. Check if the sensors found are **generic** (by locating them in other devices).
3. Compare the sensors **specifications** (sampling rate, resolution, accessibility).
4. Select the **best** one.

To identify sensing components suitable for SbSCA, a set of devices having comparable specifications was evaluated. The chosen devices and their references are listed below:

- **NXP** *i.MX 6*
- **NXP** *i.MX 8*
- **ST Microelectronic** *STM32MP1*
- **Xilinx** *Zynq-7000*
- **Xilinx** *Zynq UltraScale*
- **Intel** *Arria 10*

These 6 devices were chosen because they are designed by various vendors (Xilinx, ST Microelectronics, NXP or Intel) and built for IoT, automotive and other connected applications which typically meet the type of usage potentially targeted by SbHA. Moreover, these devices implement equivalent architectures (all based on ARM Cortex-A processors) and their documentation is easily accessible. After conducting a deep analysis of

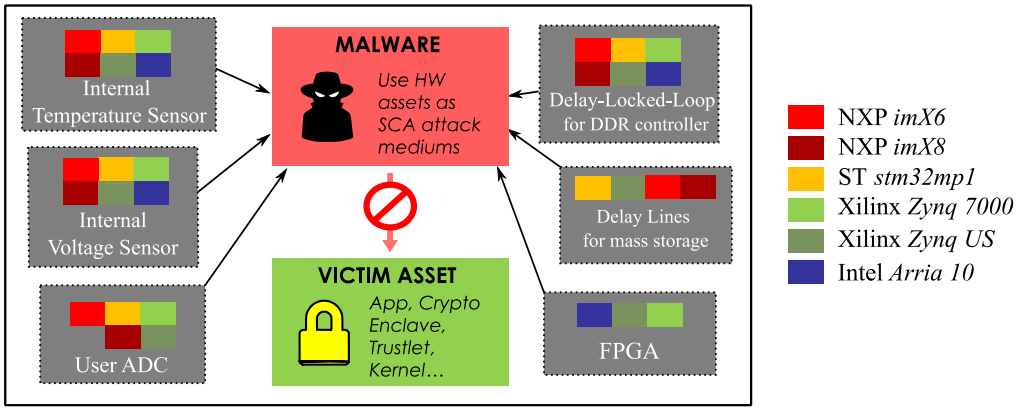


Figure 4.1 Benchmarking the resources available for voltage sensing

their underlying hardware, we were able to identify various internal components exposing a relationship with on-chip voltage fluctuations.

Six hardware components potentially suitable for power Side-Channel Analysis (SCA) were found in the six device's data-sheets. Figure 4.1 shows which device implements which vector and thus gauges if the latter is generic. For instance, an FPGA fabric is only available on Intel and Xilinx devices while Delay-Locked-Loops (DLLs), temperature, voltage and Analog-to-Digital Converter (ADC)-based sensors are implemented in every considered SoCs. The list below details the sensor specifications retrieved from the data-sheet.

1. **Internal Voltage Sensor -** Internal voltage sensors are generic and usually found in the power management unit. They are typically used by software applications to inform the user about the actual Computer Processing Unit (CPU) power consumption. In the evaluated devices, the chip's voltage is captured using an ADC with a maximum sampling rate ranging from 500 KB/s (for the *i.MX8* board) to 1 MB/s (for the *Zynq-7000* and Ultra Scale processors). Its resolution is located around 6 (*i.MX8*) to 12 bits (*Zynq-7000*). Depending on its implementation it can either be accessed through a simple register read or require the ADC configuration to probe the power lines (*Zynq-7000* XADC).

2. **Internal Temperature Sensor -** Temperature sensors are generic and usually found in the thermal management unit. They are typically used to protect the device from overheating. The temperature is captured using either an analog sensor (e.g., thermal diode in the *i.MX8*) or a digital sensor (digital temperature sensor in the *STM32MP1*). The maximum temperature sampling rate found in these devices ranges from 200 S/s (for *Arria 10*) to 666 KS/s (for the digital *STM32MP1* sensor). From an accessibility and resolution point of view, temperature sensors provide specifications similar to those of internal voltage sensors.

3. **User ADC:** All the devices presented except the Intel *Arria 10* implement at least one user-programmable ADC. It can be configured to capture external inputs or in-

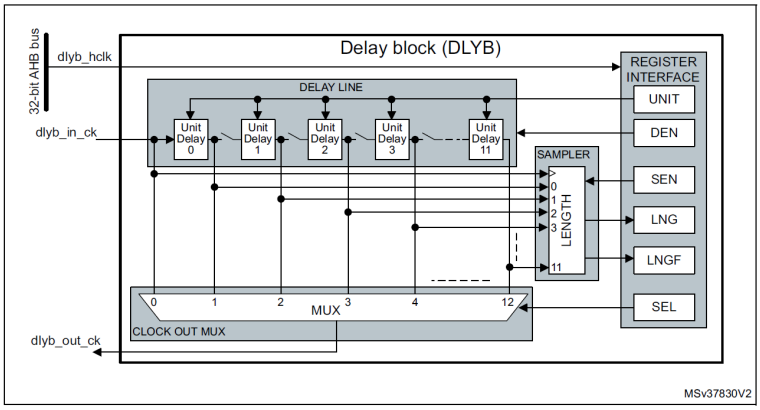


Figure 4.2 STM32MP1 *Delay Block* Diagram taken from [93]

ternal power lines for voltage sensing (e.g., *Zynq-7000* XADC). The maximum sampling rate indicated ranges from 1 MS/s (for the *Zynq-7000* SoC) to 7.2 MS/s (for the *STM32MP1* SoC). ADC resolutions range from 12 bits (e.g., *Zynq-7000*) to 16 bits (e.g., *i.MX8*) and their access necessarily requires a complex calibration to set channel, input and various ADC parameters.

4. **FPGA -** Reconfigurable logic is not generic and is even over-represented in our device list. However, as we've seen in the previous chapter, it offers a maximal sampling rate of 250 MS/s and a resolution of 6 bit for the Time-to-Digital Converter (TDC)-based sensor. For this reason, it is by far the most efficient method for sensing SoC leakage internally.

5. **Delay-Locked-Loop -** Subsequent researches highlighted the systematic presence of DLL components within the evaluated devices. Located in the Double Data Rate (DDR) memory controllers, DLLs are used for synchronizing clock and data lines during external memory transfers. They are dynamically configured according to Process Voltage Temperature (PVT) variations to ensure proper communication.

   These elements are quite close to the TDC sensors used in Chapter 3 since they are mainly made out of delay-lines. Even if DLLs are not directly sensors, we believe that these entities could be diverted from their main purpose to obtain information about the actual on-chip voltage.

   DLLs can be easily tested since they are memory mapped and readable through a simple register access. Based on three initial tests conducted on the *Zynq-7000*, the *STM32MP1* and the *i.MX6* boards, a DLL access rate of 16 MS/s was obtained. Their resolution can be approximated by taking the number of distinct delay values that can be applied to the DLL. For instance, in the *Zynq-7000* SoC, 256 delay values can be programmed. This gives an overall DLL sensing resolution of 8-bits. Section 4.2 describes how DLL works and how a voltage information can be extracted from the delay value.

6. **Delay-Line Block -** Additionally to the DLLs, we also observed the adoption of

Delay-Line Blocks (DLBs) in mass storage memory controllers (especially in the most recent devices benchmarked). While the use of DLLs was first reserved to high-bandwidth DDR memories, the continuous increase of Non-Volatile Memory (NVM) speed is forcing designers to adopt delay-lines for compensating PVT variations. The DLBs observed in the tested devices are dedicated to external SD, MMC cards (e.g., *i.MX8* and *STM32MP1*) and Quad SPI Flash interfaces (*STM32MP1* and *Zynq Ultra Scale*). These DLBs are basically TDCs implemented within ASIC devices. The DLB diagram taken from the *STM32MP1* documentation [93] and illustrated in Figure 4.2 is a perfect example. It contains a complete delay-line plus a sampler deployed to calibrate the delay-line. This sampler is exactly equivalent to the flip-flop registers that were used in Chapter 3 to capture the TDC-based sensor delay-line state in an FPGA.

Based on a test conducted on the *STM32MP1* DLB we obtained a sampling rate of around 15,2 MS/s for the DLB. Its resolution can be taken as the number of distinct delay values implemented. For instance, in the *STM32MP1* SoC, 128 delay values can be programmed. This gives a total resolution of 7-bits. Section 4.2 further describes how DLBs work and how a voltage information can be extracted from the sampler.

4.1.2 Selecting an SbSCA Vector

Sensor	Generic	Sampling Rate	Resolution	Circuit
Voltage sensor	Yes	0.5-1 MB/s	6-12 bits	Analog
Temperature sensor	Yes	0.2-666 KB/s	6-12 bits	Analog
User ADC	Yes	1-7.2 MB/s	12-16 bits	Analog
FPGA-based sensor	No	250 MB/s	6 bits (TDC)	Digital
DLL	Yes	16 MB/s	8 bits	Digital
DLB	Yes/No	15.2 MB/s	7 bits	Digital

Table 4.1 On-chip SCA vector benchmark results.

Table 4.1 provides an overview of the different SCA vectors found and their characteristics. To make the attack reproducible from a target to another we need the vector to be generic, that is, implemented in a large variety of devices. For this reason, FPGA-based power analysis cannot be selected since reconfigurable logic is generally not implemented in today's SoCs.

All the other sensing mediums found fulfill the genericity requirement. However, they provide heterogeneous performances. For instance, the temperature sensors are limited in sampling frequency and as mentioned in Chapter 2, they are not suitable for advanced SCA attacks. The remaining vectors can be classified as analog-based and digital-based sensing mediums. Analog sensors based on ADCs suffer from a limited sampling rate related to conversion time. The best solution for maximizing sampling rate is thus the digital option based on delay-lines. For its part, the resolution cannot be evaluated as

a reliable parameter since the voltage resolution is not mentioned (not found for delay-lines). As a reminder, the voltage resolution defines the sensor's granularity. If the power supply voltage fluctuation is smaller than the sensor's voltage resolution, the variation cannot be captured. Because the power supply fluctuations are tiny, a large part of the overall resolution won't be used. For instance, our experimentation on delay-lines showed that voltage drops only affected 10 to 15 delay levels over the 128 available (*STM32MP1*).

Considering the benchmark conclusions and the fact that delay-lines had never been studied as SbSCA sensors in any research work, we decided to direct our work toward these unsuspected sensing components. Our suspicions on their capability to collect SCA leakage were rapidly validated and led to the assembly of various SbSCA scenarios. With the identification of delay-lines in SoCs, a digital, fast and generic sensor was added to the SbSCA vector list. The various studies conducted using delay-lines are described in the remainder of this chapter.

4.1.3 Chapter Outline

The remainder of this chapter is organized as follows

- Section 4.2 is dedicated to the technical description of delay-lines-based components implemented in complex devices.
- Section 4.3 describes inter-core power SbSCA attacks conducted on complex SoCs using delay-lines.
- Section 4.4 is dedicated to additional delay-line applications such as covert-channels and public key SCA attacks.
- Finally, Section 4.5 discusses the results obtained and concludes this chapter.

4.2 Technical Background

Delay-line-based sensors were previously used in FPGA devices as a way to monitor chip power consumption (TDC sensor). Despite offering great performance, these sensors were limited to configurable logic which is rarely integrated in SoC devices. In this section, we disclose that digital and analog delay-lines are widely implemented in SoC memory controllers. We present them and discuss their potential use as voltage sensors (delay sensors).

4.2.1 Memory Controller Basics

Because high-end SoCs are designed to run complex Operating Systems (OSs) (Linux, Android, etc.), they require a large amount of NVM to store the OS and Random-Access-Memory (RAM) to efficiently load it. Due to technological constraints, these SoCs do not embed a significant amount of RAM nor NVM memory but are rather interconnected with external memories (memory cards, Flash memory, Synchronous Dynamic Random-Access Memory (SDRAM), etc). Thus, depending on the form-factor, speed and memory size requirements, designers can choose between a wide range of external memory devices. A typical scenario of a SoC using external memories is depicted in Figure 4.3.

Several memory controllers are required to interface the SoC with its external memories. Each memory controller acts as a request arbiter, a transaction scheduler and as a physical interface to manage data flowing from the SoC to the memory, and vice-versa. In embedded systems, for cost and efficiency reasons, the memory controller is more likely to be directly integrated as a part of the SoC. At the edge of the memory controller, a physical controller (dotted lines in Figure 4.3) outputs and captures the signals that will flow between the SoC I/Os and the memory chip I/Os (clock, data, configuration signals, etc.). The physical controller also ensures that these signals arrive on time regardless of the interconnection tracks length on the Printed Circuit Board (PCB) and the PVT variations. To better understand the extent of memory signal propagation timings, we draw a

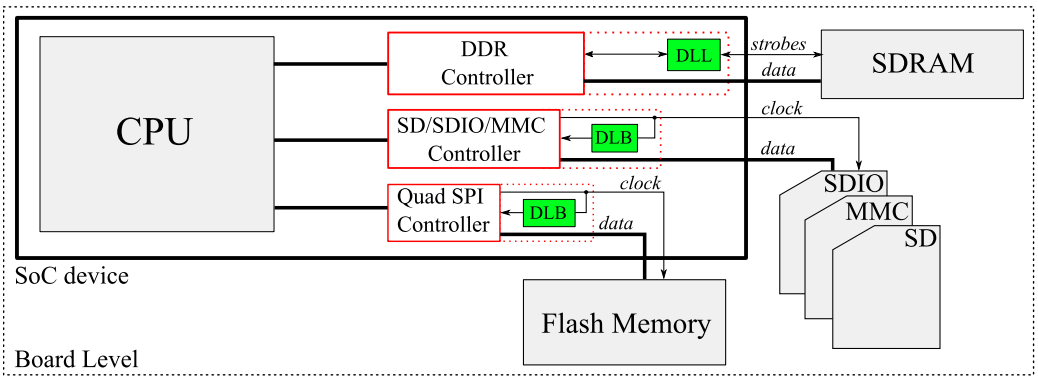


Figure 4.3 Typical SoC connectivity with external memories. Delay-lines are implemented to synchronize clock and data signals arrival in the memory controllers.

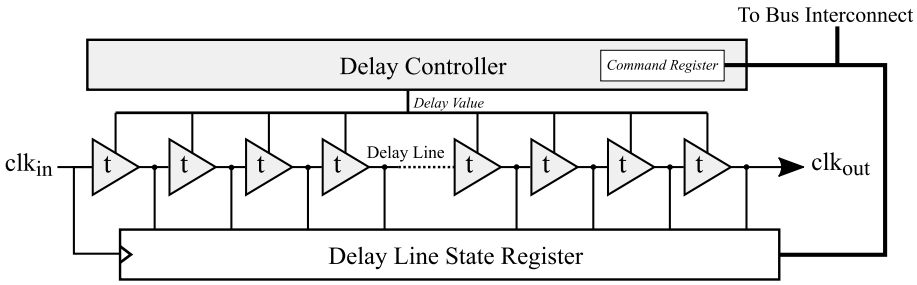


Figure 4.4 An example of DLB used in low-bandwidth memory controllers.

simple example of SDRAM association.

When a read operation is initiated by the SoC, the external SDRAM memory outputs the requested data edge-aligned with a clock signal (strobe) later dedicated to data sampling. Depending on the PCB tracks length, the clock signal is likely to shift ahead of the data signals, leading then to a sampling error. To mitigate this effect, the SoC physical controller implements delay-line-based components (DLL and DLB in Figure 4.3) to calibrate the phase alignment between the sampling clock and the data signals. This calibration can be manual and made once and for all after testing at manufacturing or performed at each chip power-up. It can also be adjusted dynamically to counterbalance any misalignment due to power supply or temperature fluctuations.

The relationship between the delay applied and the SoC voltage fluctuations drew our interest. In the following paragraphs, we present two different delay-line-based mechanisms that can be used to generate these delays for low and high-bandwidth external memory applications.

4.2.2 Delay-Line Blocks in Low-Bandwidth Memory Controllers

In relatively low-bandwidth external memories such as Flash memories, SD cards and multimedia cards, the impact of voltage and temperature fluctuations is considered not significant enough to jeopardize the communication integrity: dynamic calibration is not required. Delay-lines are nonetheless used to mitigate the impact of the PCB track length on the data and clock signals propagation timings (these delays are not predictable by SoC designers, they are set only at board design time). As track lengths are fixed, a static delay is sufficient to ensure good operation. For a read transaction, the delay-line is typically calibrated in order to add a phase shift of 90° to the clock signal. Thus, it ensures that data signals are in place when sampling occurs. The delay-line calibration is carried out through a series of training steps. These training steps modify the delay of the elements forming the chain and, for each configuration, verify if the external memory has been properly read. If the training is successful, the delay-line configuration is saved in a dedicated register and remains unchanged until the next test.

Several SoC vendors provide user programmable DLBs as a way for developers to

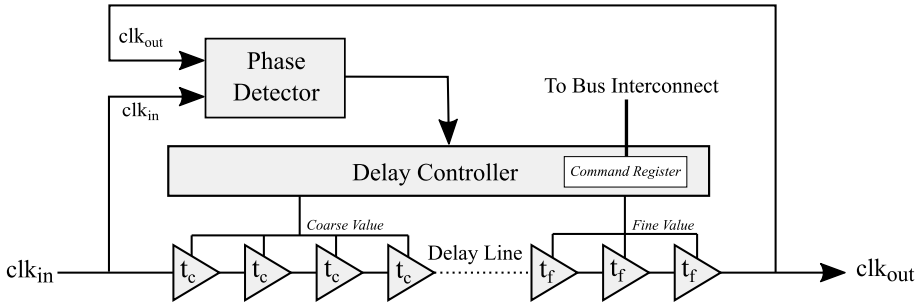


Figure 4.5 An example of DLL used in DDR memory controllers.

be able to use a wide range of memory chips or cards with different bus speeds. Unlike traditional static delay-lines, these DLBs come with both a complete calibration toolkit and a detailed documentation. Figure 4.4 illustrates the DLB structure that was observed in one of the SoC we benchmarked. Its purpose is to delay the clock signal with respect to the data signals when a read operation is conducted. The DLB consists in a simple delay-line associated with a set of control and status registers. A *Command Register* controls the delay t of all the delay-line elements and thus the phase shift added to the clk signal. To ensure that the phase shift obtained is conform to the applied command, a *state register* captures the output of each element forming the delay-line every time a clk_{in} rising edge event occurs. Then, a specific training is performed to verify whether the captured pattern matches the command or not.

Despite some missing parts, this structure is reminiscent of that of a TDC as the delay-line state is continuously captured and stored in an accessible register. In Section 4.3.4, we demonstrate that this DLB can be turned into a voltage sensor and hijacked to perform a power SCA.

4.2.3 DLLs in High-Bandwidth Memory Controllers

Because of the continuous increase of memory bus speeds, the available slack time for data sampling is gradually shrinking. DDR memories such as SDRAM memory perform one data transfer per clock edge (both rising and falling) while reaching gigahertz frequencies [117]. On these devices, the data sampling is very likely to get corrupted by temperature and voltage variations. This time, a static delay source is not suitable to ensure correct operations. To effectively cancel voltage and temperature noise side-effects, a dynamic way to adapt the clock delay has to be considered.

DLLs are generally used in recent DDR memory controllers to dynamically track and control the phase shift applied between the sampling clock and the external memory (e.g., SDRAM) data signals [7, 25]. As illustrated in Figure 4.5, a DLL has two main blocks: a delay-line, and a feedback circuit. The delay-line is calibrated to provide a phase shift to a *clk* signal using both *coarse* and *fine* delay elements. However, the propagation delay jitter associated with on-chip voltage and temperature fluctuations is likely to skew the applied phase. This is why a DLL includes a feedback circuit to tune the delay-line in

order to provide a dynamic control of the phase shift and thus, counterbalance voltage and temperature variations. The feedback circuit comes with a phase detector that compares the phase shift between the clock signal at the input of the delay-line, clk_{in}, and its phase-shifted clock output, clk_{out}. Then, according to the measured error, a delay controller applies a correction in order to "deskew" the result, that is, to get back to the initial delay. The applied correction modifies the delay of the elements forming the delay-line and can be either analog or digital-controlled depending on the delay-line type [1].

A *command register* stores the delay settings, it is memory-mapped and hence can be read from the SoC Application Processor (AP) or Microcontroller Unit (MCU) cores. The DLL operates autonomously, this means that through a simple access to this register, a process can retrieve the state of the DLL, which shall be correlated to on-chip voltage and temperature variations. As a result, tracking the *command register* content shall provide an image of the SoC power consumption that may be used to carry out SCAs. Note that this measurement methodology (tracking the command of a feedback dynamically controlled system) differs from that described in Section 4.2.2 for DLBs (sampling a clock signal propagating inside a fixed delay-line). If this unusual measurement medium provides enough resolution and sampling rate to eavesdrop power consumption of secure applications running on a processor, this could represent an important backdoor for computer security. This hypothetical vulnerability is strengthened by the fact voltage sampling only requires a read access to the command register, no configuration steps are required. A DLL-based power SCA attack scenario is developed in Section 4.3.3.

4.3 SideLine: Delay-Line-based power SCA on complex SoCs

4.3.1 Introduction

In this section we describe *SideLine*, a novel SbSCA vector based on the intentional misuse of hardware resources available in high-end SoC devices. *SideLine* leverages delay-lines components embedded in SoCs that use external memory; it neither requires embedded reconfigurable logic (FPGA) nor analog circuitry (ADC). Two delay-line-based blocks namely DLL and DLB are hijacked to perform voltage measurements and maliciously used to conduct power SCAs on AP and MCU. *SideLine* makes it possible for an attacker to perform SbSCA without direct physical access to the target. Our contributions are listed below:

- We describe three attacker-victim (core-vs-core) delay-line-based SCA scenarios over two SoC devices: **AP-vs-AP** attack (on a Xilinx *Zynq-7000* SoC), **AP-vs-MCU** attack and **MCU-vs-AP** attack (on a STMicroelectronics *STM32MP1* SoC) where AP and MCU respectively denote the application processor and the microcontroller cores.
- For each scenario a correlation power analysis attack is conducted against the publicly available OpenSSL Advanced Encryption Standard (AES) encryption algorithm [30] and the full secret key is successfully recovered. The attack feasibility is demonstrated on bare metal and Linux OS-based applications.

Outline: The remainder of this work is organized as follows. In Subsection 4.3.2, we present the tested products and the associated threat model. Subsections 4.3.3 and 4.3.4 are dedicated to the deployment of the three attack scenarios. Finally, we discuss performance, limitations, countermeasures in Subsection 4.3.5 and conclude in Subsection 4.3.6.

4.3.2 Experimental Setup

4.3.2.1 Tested Devices

Two devices from two different SoC providers have been studied in our experiments. The first target considered in this work is a Xilinx *Zynq-7000* SoC [141] that comes with a dual-core Cortex-A9 AP. It is a typical multi-purpose SoC providing many additional resources: FPGA, I/O, ADCs, bus controllers, etc. It supports DDR2-DDR3, Flash and SD/MMC external memories and provides several DLL blocks to interface properly with DDR external memories. The experiments made on this target have been conducted without using an OS: we denote it as a **bare metal attack**. This configuration makes SCA easier as there are fewer interruptions (with respect to the case in which

an OS is used) that may disturb the attack and victim processes and cause synchronization issues. The entire *Zynq*-based *SideLine* attack code can be cloned from GitHub: `https://github.com/Remote-HWA/SideLine_Zynq`.

The second target is a STMicroelectronics STM32MP157C-DK2 development board [93] that embeds a dual-core Cortex-A7 AP associated with a Cortex-M processor (MCU). It also supports DDR2-DDR3, Flash and SD/MMC external memories and embeds several DLL blocks. Additionally, it provides user programmable DLBs (DLYB in [93]) that can be employed for interfacing low bandwidth memory (e.g., an SD card). These programmable DLBs are the second case we studied. The experiments done on this SoC have been carried out with a Linux OS running on its AP (i.e. the Cortex-A7 processor). The results are those of a **Linux OS attack**. The entire *STM32MP1*-based *SideLine* attack code can be cloned from GitHub: `https://github.com/Remote-HWA/SideLine_STM32MP1`.

4.3.2.2 OpenSSL AES Architecture

The OpenSSL library [106] provides several cryptographic algorithms used for securing channels over computer networks. In this work, we focus on the OpenSSL AES-128 (version 1.1.1) that implements a 32-bit tabulated version of the textbook AES encryption algorithm [30]. This variant merges the `Mixcolumn` and `SubBytes` transformations into 4 pre-computed look-up tables known as T-tables (256 x 32-bit) as a way to optimize the computations on 32-bit processors.

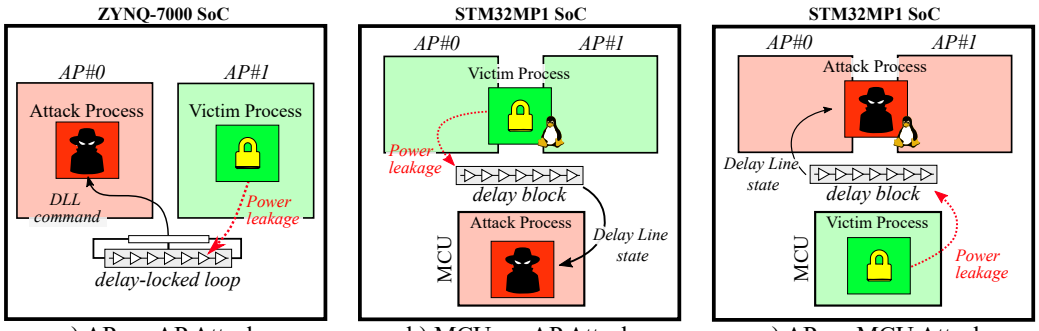
4.3.2.3 Threat Model

In this section, we introduce three core-vs-core attack scenarios in order to assess the SCA capabilities of the delay-line-based sensors. For each scenario depicted in Figure 4.6, we first deploy a cryptographic application (in green) within a processor core. This application located either in the AP or in the MCU allows the end-user to launch AES encryptions/decryptions, with the plaintexts/ciphertexts that he provides. Secondly, we introduce a malicious user (in red) that has the privilege level necessary to access the DLBs presented in Section 4.2.2 and that uses them to retrieve the leakage induced by the AES application.

Although not used in this research work, Trusted Execution Environments (TEEs) based on the TrustZone (TZ) [7] architecture stand as potential realistic targets for the delay-lines. TZ attacks from the normal-world to the secure-world have been widely covered in recent remote attack works [128, 111, 20, 104]. However, from a side-channel point of view, the current TZ does not provide any countermeasures. Thus, the ability of an attacker to turn our feasibility attack into an end-to-end TZ attack is reasonably expected.

In the remainder of the section, the three scenarios presented are referred to as:

1. A **DLL-based attack** (Figure 4.6.a), or AP-vs-AP attack, that demonstrates the



a) AP-vs-AP Attack b) MCU-vs-AP Attack c) AP-vs-MCU Attack

Figure 4.6 Basic principle of the three core-vs-core attack variants proposed in this work. It shows the leakage path from the victim process to the delay sensor and the sensor data flow retrieved by the attack process.

ability of a DLL to serve as a power supply sensor suitable for a Correlation Power Analysis (CPA) attack against the AES algorithm. In this scenario, one core of the *Zynq-7000* processor runs the AES victim application, while the second core executes the attack process (both victim and aggressor processes are C programs, in bare metal mode). The attacker code is in charge of collecting the leakage data of the AES. It does so by configuring the access to the DLL command register that makes it possible to sample its values during AES encryptions performed by the first core. The attacker core is also in charge of providing the plaintext to be ciphered by the victim process and to trigger both the encryption and readback of DLL states. This AP-vs-AP attack scenario is described in details in Subsection 4.3.3.

2. A first **DLB-based attack** (Figure 4.6.b), or MCU-vs-AP attack, where the victim process is ran on the *STM32MP1* AP (a C code AES running on top of a Linux OS) and the attack process is executed by the Cortex-M MCU (a C program, in bare metal mode). In this scenario the MCU is in charge of calibrating and using a DLB to eavesdrop the activity of the AP. This MCU-vs-AP attack scenario is addressed in Subsection 4.3.4.

3. A second **DLB-based attack** (Figure 4.6.c), or AP-vs-MCU attack, that matches a typical state-of-the-art industrial case where the cryptographic and security operations of a SoC embedding AP cores are delegated to a less complex MCU core. In this scenario the AP core (Cortex-A7) runs the attack process while the MCU core (Cortex-M) runs the AES victim process. This AP-vs-MCU attack scenario is reported in Subsection 4.3.4.

4.3.3 DLL-based Power Side-Channel Attack

This subsection presents a novel way to monitor on-chip voltage fluctuations and conduct power SCAs using the DLLs embedded in SoC memory controllers.

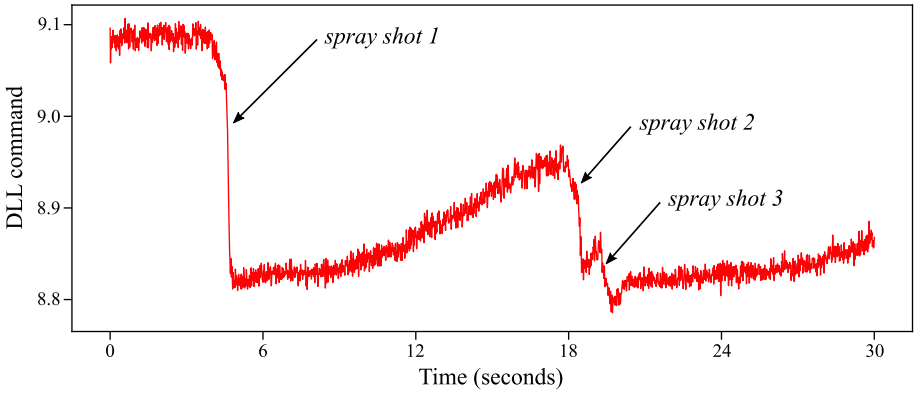


Figure 4.7 DLL response to sudden temperature drops induced by three successive exposition of the SoC to a cooling spray.

4.3.3.1 Validating DLL Effectiveness: *Monitoring Temperature*

As a proof of concept, a simple experiment was carried-out on the *Zynq-7000* SoC to confirm that the DLL command is actually tracking the SoC package temperature variations. The test uses a C program designed to continuously read and store the DLL command register content into an acquisition array for a period of 30 seconds. Simultaneously, a cooling spray was used at specific moments to cool down the SoC package. To limit the acquisition size, each array index contains the average of 1,000 successive DLL readings. Figure 4.7 reports the evolution of the measured DLL command (y-axis) as a function of time (x-axis). Each spray shot induces a temperature drop (translated into a DLL command drop in Figure 4.7) that progressively recovers until the next one. This simple experiment confirms that a DLL is suitable to dynamically track the SoC temperature variations. As the temperature decreases, the propagation speed of the clk signal through the delay-line increases [35]. Thus, the phase-shift between clk_{in} and clk_{out} progressively drifts. To counterbalance this effect, the DLL dynamically adapts its command in order to maintain a constant phase shift. Because package temperature evolves relatively slowly, the sampling frequency for this experiment was limited to 300 kHz. However, as this chapter focuses on power side-channel, which itself depends on transient voltage drops measurements, a higher sampling rate needs to be achieved: it is the subject of the next Subsection 4.3.3.2.

4.3.3.2 Improving Sampling Rate and Synchronisation using DMA

As mentioned before, the DLL command value can be directly accessed through its memory address. Then, a loop associated with an array can be added to collect more samples. This CPU-based sampling method works in principle but has several drawbacks:

First, it requires a constant time between each acquisition. If this constant time is not achieved, the samples of different encryptions won't be correctly aligned. Consequently, statistical attacks will be less accurate as the averaging of several acquisitions will suffer from de-synchronisation. Achieving constant time is feasible in bare metal applications

because they rarely suffer from interruptions. However, if the application runs over an OS, interrupts will dramatically affect the timing of acquisitions and make their averaging impossible. The second limitation is related to the achievable sampling rate. Indeed, the delay induced by CPU memory access plus the storage of the acquired data into an array is not optimal. Using this method on the *Zynq-7000* SoC, the sampling frequency was limited to 2.2 MHz.

To solve these issues, we choose to use Direct Memory Access (DMA) in order to improve the sampling rate as well as the synchronization of our samples (as proposed in [46]). A DMA is a hardware module able to transfer data from a peripheral to another without processor intervention. For this reason, it is faster in transmitting data, but also not affected by OS interrupts. The source address (address from which the DMA should sample the data) is the register containing the DLL command. The destination address (destination of the DMA transfer) is the base address of an array whose size depends on the number of samples required. At the end of the DMA transfer, an interrupt flag is set and ends the sampling process. With DMA up and running, we improved the DLL sampling frequency from 2.2 MHz to 16 MHz.

4.3.3.3 Bare Metal OpenSSL AES Attack Setup

According to the threat model we consider (see subsection 4.3.2.3), the attack process shall be able (1) to trigger the start of an AES encryption by the victim process, and (2) to control the gathering of the leakage from the AES through a DLL-based voltage sensor. Our test bench includes two processes (their pseudo code is given in Algorithm 1) executed by the two application cores of our target in bare metal mode: the attack process on AP#0 and the victim process on AP#1.

The victim program starts by an initialization step (the AES round keys are derived from the secret key), denoted $\texttt{AES}_{init}()$, and then enters an infinite loop waiting for the inputs of the attack program (`Wait for Start`$_{\texttt{AES}}$ `flag()`). The initialization step of the attack process consists in setting the configuration of the DMA access to the DLL command register (in order to retrieve dynamically the values it contains which are correlated to the AES calculations), denoted $\texttt{DMA}_{init}()$, and in setting the serial communication with a control PC $\texttt{UART}_{init}()$ to retrieve these values and the plaintext (used for conducting a first round CPA attack). The CPA attack we carried out requires to encrypt Nb_{acq} plaintexts and to gather the associated leakage. The attack program then enters a `while` loop with Nb_{acq} iterations. Each loop consists in: (1) sending the plaintext to the victim process (`Send AES plaintext to AP#1`), (2) starting the automated DMA access to the DLL command register (`Launch DMA transfer`) to retrieve Nb_{sample} times its content, (3) trigger the AES encryption (`Send Start`$_{\texttt{AES}}$ `flag() to AP#1`), (4) wait for the current encryption and DMA access to be completed (`Wait for End`$_{\texttt{AES}}$ `flag()` and `Wait for End`$_{\texttt{DMA}}$ `flag()`), and (5) to send the obtained leakage data and the plaintext to the control PC (`Export samples through UART`). Data transfer (plaintexts, ciphertexts, flags) between both processes

Algorithm 1 Dual core Cortex-A9 attack, pseudo-algorithm

/* AP#0: attack program */
Input: Nb_{acq}, Nb_{sample}
DMA$_{init}$();
UART$_{init}$();
while Nb_{acq} has not been reached **do**
  Send AES plaintext to AP#1;
  Launch DMA transfer(Nb_{sample});
  Send $Start_{AES}$ to AP#1;
  Wait for End_{AES} flag();
  Wait for End_{DMA} flag();
  Export samples through UART;
end while

/* AP#1: victim program */
Input: AES_{key}, $AES_{plaintext}$
AES$_{init}$();
while infinity **do**
  Wait for $Start_{AES}$ flag();
  Get AP#0 plaintext;
  OpenSSL AES encrypt();
  Send End_{AES} flag to AP#0;
  Send AES ciphertext to AP#0;
end while

are done through a shared memory space in RAM. On the victim side, the `while` loop synchronizes the AES encryptions (denoted `OpenSSL AES encrypt()`) with the request of a novel encryption and the delivery of a novel plaintext (`Wait for Start`$_{\text{AES}}$ `flag()` and `Get AP#0 plaintext`) by the attack process. It also signals the end of the encryption (`Send End`$_{\text{AES}}$ `flag to AP#0`) and provides the obtained ciphertext (`Send AES ciphertext to AP#0`) to the attack program.

In addition to this attack setup, we used embedded Hardware Performance Counterss (HPCs) to precisely measure the duration of an AES encryption. In average, an encryption took 837 AP clock cycles or 1,25 μs at a frequency of 667 MHz (both attack and victim programs were compiled with the optimization parameter set to -O2). The DMA transfer method we used provides a constant 62.5 ns sampling period (i.e. a 16 MHz sampling frequency). As a result, 21 samples of the DLL command are gathered per AES encryption.

4.3.3.4 DLL-based SCA Attack on *Zynq-7000* SoC

The bottom part of Figure 4.8 illustrates the results of two experiments conducted to assess the AES encryption impact on the DLL command value and precisely detect its encryption time window. The two traces depicted in black (1^{st} case) and red (2^{nd} case) represent the averaged DLL command value (y-axis) obtained for 1,000 acquisitions as a function of time (expressed in DMA samples). For the first experiment (in black), the victim program was kept idle during the entirety of the DMA sampling operations. The DLL command drop visible between sample 0 and 1,000 was induced by the extra power consumption linked to the DMA module activation. The DLL applied a strong correction to maintain a constant phase shift, that was finally relaxed as the power consumption returned to normal (from sample 2,000 to the end of sampling). The second case (in red) reports an actual iteration of the attack and victim processes when an AES encryption is done. The red trace experienced the same DLL command undershoot due to DMA

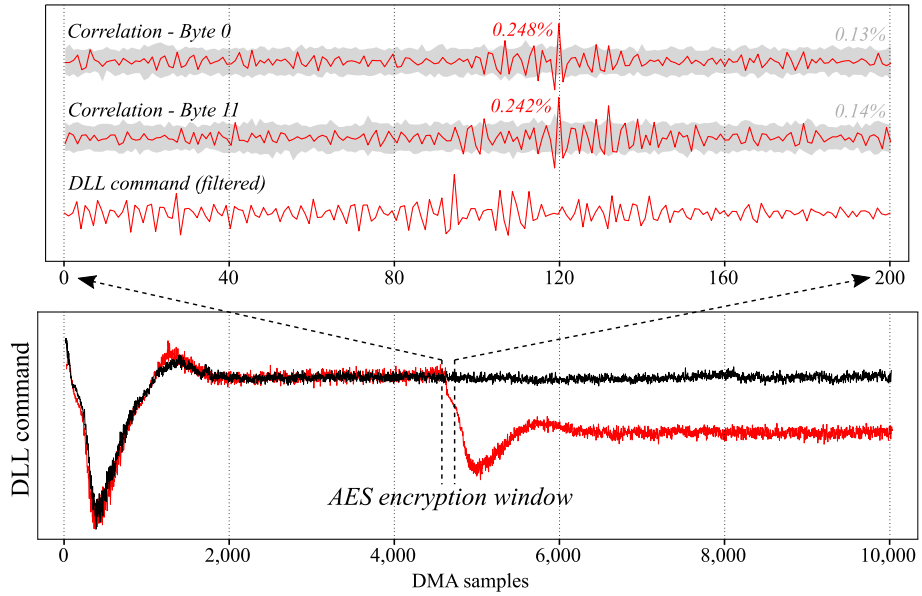


Figure 4.8 DLL-based attack results: the bottom part represents the impact of an AES encryption on the DLL command value. The top part zooms on the AES encryption windows and provides the temporal correlation rate for two key bytes.

module activation (sample 0 to 1,000) but also a second undershoot corresponding to the AES encryption (starting at sample 4,500). It is finally restored to a steady value lower than the initial one (sample 6,000 to the end of sampling). The AES encryption window was deduced from the position of the second DLL command drop. Based on this information the CPA attack could be conducted on a smaller amount of samples.

We launched a total number of 20 million AES encryptions and acquired 200 DLL command samples per encryption. Samples and plaintexts extraction through UART took around 8 hours at 921,600 bauds. Then, an external computer was used to apply post-processing to the traces and conduct the CPA attack. The top part of Figure 4.8 depicts a filtered and averaged trace of the DLL command (in red). High-pass filtering was used as a way to reduce the impact of low frequency variations (induced for instance by temperature fluctuations) on the acquired traces and thus to reduce the number of traces required for the attack. Then, we performed a plaintext-based CPA attack on the first round of the AES. As we mentioned earlier the OpenSSL AES uses T-tables to upgrade its performances on 32-bit processors. This allows us to leverage a 32-bit T-tables output prediction: $HW[T_{table}(key \oplus plaintext)]$. The obtained correlation results versus the time are represented above the averaged trace in Figure 4.8 (for two key bytes). The correct key hypotheses are depicted in red and emerge from the incorrect hypotheses (in grey) at sample 120. Based on 20 million encryptions, we achieved a full AES key recovery. 3 bytes were retrieved in the range 0-5M traces, 2 between 5-10M million, 5 between 10-15M an 4 between 15-20M. The key bytes number 7 and 9 never completely emerged from the incorrect candidates, but we assume that a simple brute force can be conducted to retrieve their values. The progressive correlation of the first 8 key bytes plus the failed byte #9 are depicted in Figure 4.20 in the appendix.

4.3.3.5 Conclusion on DLL-based SCA

In this subsection, we demonstrated that a DLL can be used to monitor on-chip temperature and power supply fluctuations. This unconventional voltage sensor was then used to conduct a power SCA on an OpenSSL AES algorithm implemented in the *Zynq-7000* application processor and a full AES key recovery was achieved (with the help of brute force for the two remaining bytes). Performance, limitations and potential countermeasures regarding this attack are discussed in Subsection 4.3.5.

4.3.4 Delay-Block-based Power Side-Channel Attack

The DLL-based attack presented in Subsection 4.3.3 was associated with the use of DDR external memories such as SDRAM in AP-based SoC. This Subsection discloses a second attack path that allows the hijacking of a programmable DLB and its malicious use to perform core-vs-core power SCAs. These experiments are conducted on the *STM32MP1* SoC.

4.3.4.1 From Delay-Block to TDC Sensor

The *STM32MP1* SoC comes with three programmable DLB Intellectual Propertys (IPs) [93] able to work with different types of external memories (QSPI, SD, MMC). Their settings can be adjusted depending on the bus speeds of the external memories used. Their initial purpose is to adjust the phase of the clock signal in order to ensure a reliable exchange of data by tuning the clock delay.

Figure 4.9 depicts the 12 elements delay-line provided by the *STM32MP1* DLB and the capture register designed to monitor the state of the output nodes of every delay element. When a clk_{in} rising edge occurs, the capture register takes a snapshot of the delay-line. This snapshot contains an image (represented as a waveform in Figure 4.9) of the clock propagation through the delay-line. The propagation delay *t* of the elementary delay elements can be set using a dedicated register. If this delay is set to its minimum the delay-line width (acquisition window) is small. Thus, only a part of the clock signal can be captured. By gradually increasing t, the clock signal observation can be extended, possibly to several periods.

We leveraged this t parameter to make the DLB sensitive to on-chip voltage fluctuations. To that end, we took a significant number of delay-line snapshots for each of the 128 possible t delay values. A vast majority of them gave stable results; which means that the captured image remained stable over successive register readings. For a few, however, delay variations arose between subsequent captures. This interesting behavior can be explained by (1) on-chip voltage fluctuations that affect the clock propagation time through the delay elements, and (2) by the fact that several delay values *t* naturally position the clock edges in unstable places within the delay line (i.e. in between two delay elements). Figure 4.9 displays three waveforms (delay-line snapshots) obtained with such a t set-

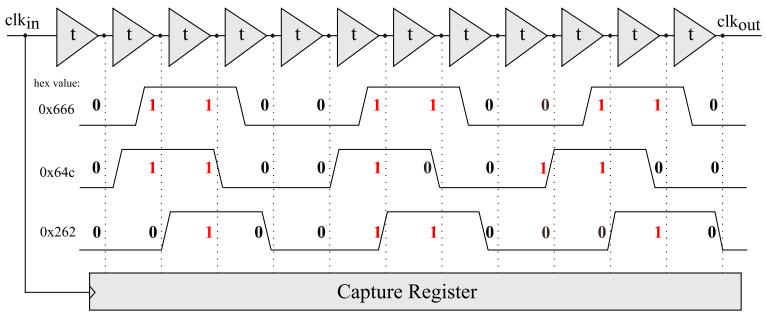


Figure 4.9 Effect of on-chip voltage variations on the sampled delay values.

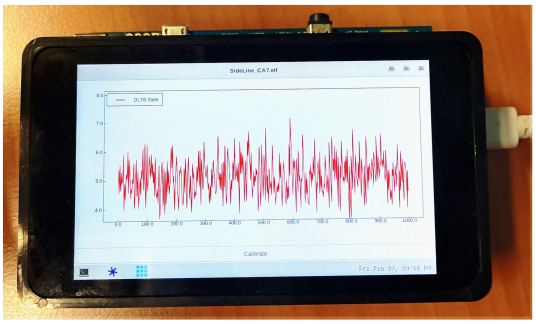


Figure 4.10 Sampled delay values displayed on screen.

ting. In this configuration, three clock periods stand in the entire delay line. From top to bottom we have: (1) the steady state register waveform which stands as our reference (it outputs a 0x666 reference value), (2) a slowed down waveform that can be obtained due to a supply voltage decrease (it outputs a 0x64c), and (3) an accelerated waveform that can be obtained due to a supply voltage increase (it outputs a 0x262). In our experiments, the three obtained hexadecimal digits are weighted and added to translate into an image of the voltage supply.

In Figure 4.10, a program displays the actual delay-line state as a function of time (as an oscilloscope) on the *STM32MP1* touchscreen. This way, the actual power consumption fluctuations affecting the DLB state can be directly observed. To make it possible, the implemented program automatically calibrates the DLB by testing various delay parameters. For each delay value, it collects multiple delay-line state samples, computes their variance and adopts the calibration that provided the highest variance. Indeed, a higher variance indicates an important delay instability and thus a stronger relationship with voltage fluctuations.

4.3.4.2 Validating Delay-Block Effectiveness: `strcmp` test

To validate the DLB effectiveness as a voltage measurement unit, we ran a simple program in the SoC Cortex-M core that induces a programmable level of core activity, thus creating different levels of power consumption. It successively alternates between low-power demanding empty `for` loops and high-power demanding string comparison functions: `strcmp`. The number of characters for the string comparison and the number of

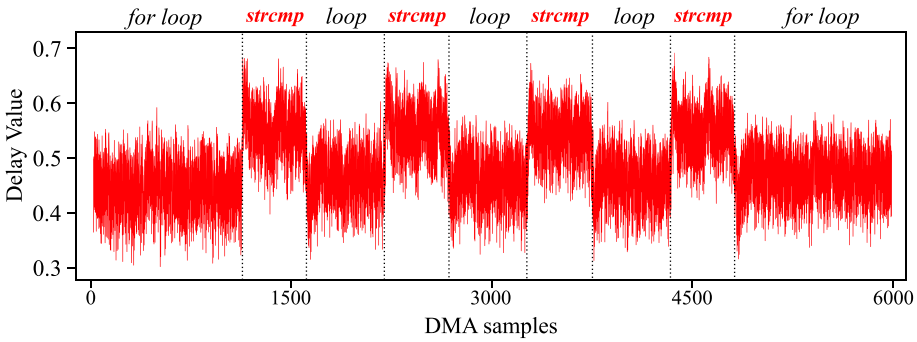


Figure 4.11 DLB response to sudden processor activity increases induced by `strcmp` computations.

increments in the loop were chosen so that they roughly take the same amount of time.

At the same time, the program uses the SoC integrated DMA in burst mode to sample the DLB capture register. On this device we identified that a single 32-bit DMA memory transfer from the DLB to the SDRAM takes around 65.8 ns, thus we obtained a 15.2 MHz sampling rate. Figure 4.11 displays the averaged delay value (y-axis) obtained for 5,000 acquisitions as a function of time (expressed in DMA samples). When the program moves from empty `for` loops to `strcmp` functions (and vice-versa), a clear delay shift appears. The `strcmp` operation generates a voltage drop, that is translated into a delay increase. It goes back to a lower level during the `for` loops. The modulation between two delay values has been induced by two different power consumption levels. This proves the relationship between the captured delay value and the on-chip activity. This validates the ability of the DLB to monitor the SoC power consumption variations (note that we could also have used the temperature monitoring technique introduced in Subsection 4.3.3.1). The following subsections describe how it can be used to conduct a CPA attack.

4.3.4.3 Linux-based OpenSSL AES Attack Setup

Similarly to the attack setup described in Subsection 4.3.3, we used the OpenSSL AES implementation to evaluate the threat posed by DLB-based SCAs. The *STM32MP1* embeds both a dual core AP and a MCU that makes it possible to test the MCU-vs-AP and AP-vs-MCU attack scenarios introduced in Subsection 4.3.2.3. Depending on the scenario, the attack and victim processes were ran either on the AP core or on the MCU core. Here, we consider the MCU-vs-AP attack to describe our attack setup.

We use an adapted version of the *Zynq*-based attack. On the adversary's side (here the MCU), DLB calibration and use of HPCs were added to the initial algorithm. HPCs are used to accurately time the successive encryptions and to mitigate the desynchronization brought by the Linux OS. For each acquisition, the number of cycles elapsed during the encryption is compared to a maximal limit Nb_{cycle} set by the adversary above which the entire acquisition is discarded. Prior to the attack, a preliminary test was conducted in order to identify the optimal value for Nb_{cycle} (assuming that a lower number of clock cycles corresponds to a lower number of interrupts). Hence, by launching thousands of

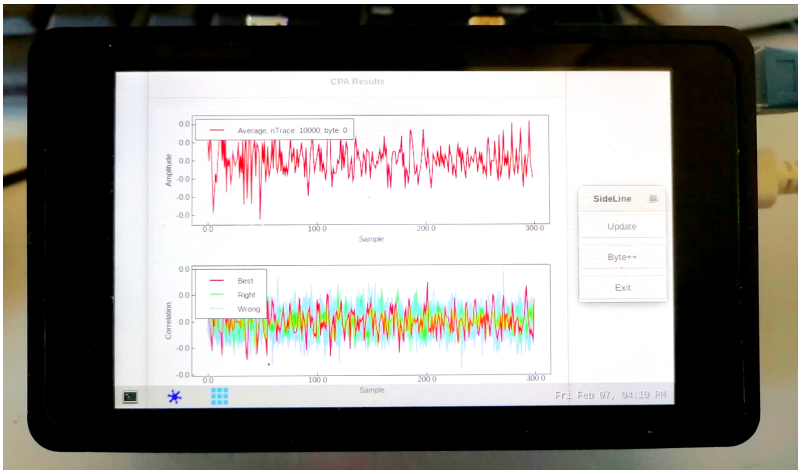


Figure 4.12 AES traces acquisition, CPA computation and GTK display (implemented for demonstration) are all embedded in the same application running within the STM32MP157-DK2 board.

AES encryptions, we were able to find a reference number of clock cycles for almost interrupt-free encryptions. Then, based on this reference, we set a maximal limit Nb_{cycle} beyond which we decided to discard the acquisitions. By doing so, at least half of the total acquisitions were retained and used for the subsequent CPA calculations.

Regarding the CPA, we embedded it directly within the *STM32MP1*. This way, we drastically limited the amount of data exported. Moreover, this allowed us to directly plot the results on screen as illustrated in Figure 4.12.

4.3.4.4 Delay-block-based SCA Attacks on STM32MP1 SoC

In the AP-vs-MCU attack scenario, the OpenSSL AES program runs within the *STM32MP1* Cortex-M MCU. Using compiler optimization set to -O0, 1,460 clock cycles are required to perform a single AES encryption, that is 7.3 μs at the MCU operating frequency (200 MHz). Figure 4.13 displays in its bottom part the averaged delay values obtained for a time window of 250 DMA samples (or 16.4 μs) over 10 million acquisitions. The AES encryption, which approximately covers 110 DMA samples, is surrounded by two empty `for` loops added for visualisation ease. The top part of Figure 4.13 provides the CPA correlation rates of four key bytes (of index #1, #13, #9, and #5) as a function of time. The correct key hypotheses are depicted in red and emerge from the incorrect hypotheses (in grey) between samples 70 and 80. We chose to represent these key bytes because they are equally distant regarding the OpenSSL byte computation order: *0* ***5*** *10 15 - 4* ***9*** *14 3 - 8* ***13*** *2 7 - 12* ***1*** *6 11*. This explains the regular temporal offset observed between them. Based on 10 million encryptions, we achieved a full AES key recovery. 6 bytes were retrieved in the range 0-2M traces, 4 between 2-6M and 6 between 6-10M. The progressive correlation of the eight last AES key bytes (#8 to #15) are depicted in Figure 4.21 in the appendix.

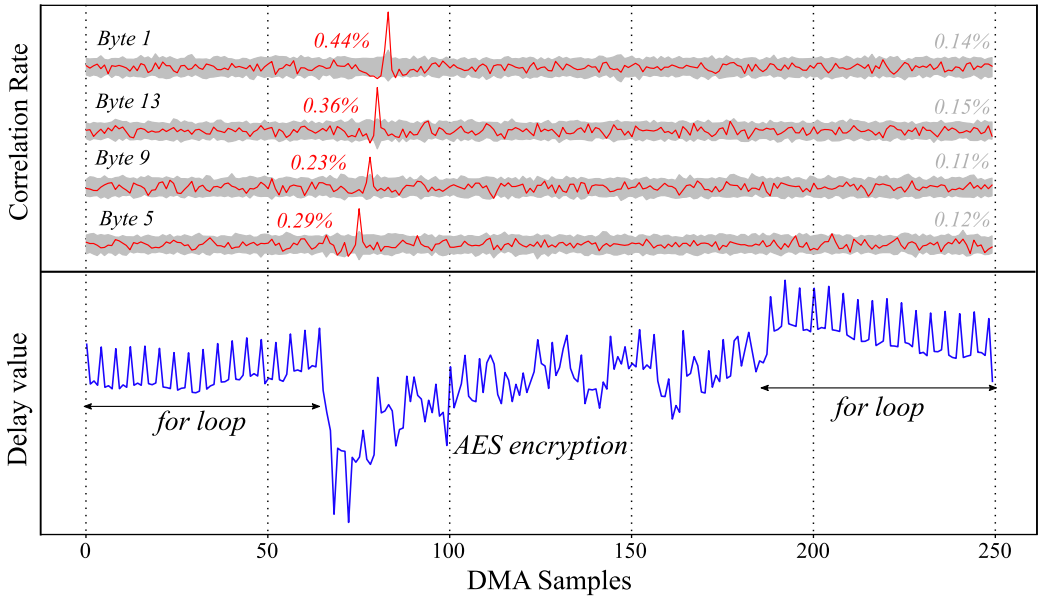


Figure 4.13 AP-vs-MCU attack results: the bottom part represents the averaged AES power consumption, the top part provides the correlation rates as a function of time for four AES key bytes.

In the MCU-vs-AP attack scenario, the OpenSSL AES program runs in the *STM32MP1* Cortex-A7 AP. Using compiler optimization set to -O2, 865 clock cycles are required to perform a single AES encryption, that is 1.33 μs at the AP operating frequency (650 MHz). Figure 4.14 displays in its bottom part the averaged delay value obtained for a time window of 100 DMA samples (or 6,6 μs) over 40 million acquisitions. The AES encryption, which approximately covers 20 DMA samples, is surrounded by two empty `for` loops added for visualisation ease. The top part of Figure 4.14 provides the temporal correlation rate of four key bytes as a function of time . The correct key hypotheses are depicted in red and emerge from the incorrect hypotheses (in grey) between samples 30 and 40. Again, we chose to represent these specific key bytes because they are equally distant in the OpenSSL byte computation order. However, the AES encryption in the AP is faster than that of the MCU (1.33 μs vs. 7.3 μs) and the DMA sampling frequency that remained fixed between the two experiments is no longer sufficient to let the temporal offsets appear. This limited sampling frequency partly explains the higher number of acquisitions required to retrieve some key bytes. For instance, byte #12 in Figure 4.14, seems to suffer from the under sampling and gave poorer correlation results (0,07%) than byte #4 (0,32%) or byte #0 (0,29%). We were able to confirm this assumption through a second experiment where the AES encryption temporal window had been slightly shifted regarding the DMA: the AES leakage was thus sampled at different timings. This experiment gave better results on several key bytes that struggled to emerge in the previous attack. Based on 40 million encryptions, we achieved a full AES key recovery. 3 bytes were retrieved in the range 0-10M traces, 6 between 10-20M, 2 between 40-30M, 4 between 30-40M. The 13th key byte never completely emerged from the incorrect candidates, but we assume that a simple brute force can be conducted to retrieve

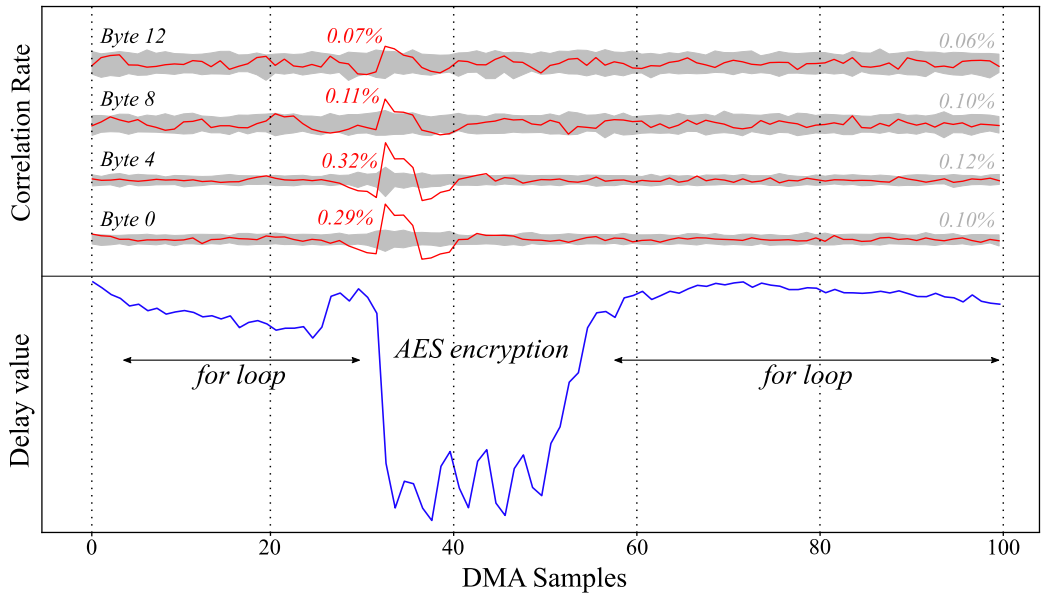


Figure 4.14 MCU-vs-AP attack results: the bottom part represents the averaged AES power consumption. The top part provides the correlation over the time results over four AES key bytes.

its value. The progressive correlation of the first key bytes (0 to 7) are depicted in Figure 4.22 in the appendix.

4.3.5 Discussion

Two delay-line-based power measurement techniques, using a DLL or a DLB were introduced and studied in this section. Because such delay-line-based components are embedded in almost every high-end digital SoC that uses external memories, the threat model we introduced is serious and shall be considered feasible for a large number of complex SoCs. In this subsection, we discuss performance, additional attack scenarios and potential countermeasures regarding the *SideLine* attack.

4.3.5.1 Performance and Limitations of *SideLine*

Table 4.2 summarizes the results obtained for the three attack scenarios considered in this chapter. First, an AP-vs-AP attack was performed on a *Zynq-7000* SoC using DLL-based sensors. As DLLs provide a limited resolution, a large amount of acquisitions were required to integrate enough information for the CPA to succeed (20 million traces required for full AES key recovery). It took around 12 hours to extract the traces, apply post-processing (filtering) and conduct the CPA attack. The lack of resolution also made post-synchronization nearly impossible and thus implied the collection of leakage traces with a constant synchronization. Apart from performances, the DLL was by far the simplest sensor to implement in our experiments, as it only required the read access to a memory-mapped register. However, care must be taken as in certain cases, DLLs

Scenario	Sensor	Nb_{Acq}	$freq_{DMA}$	$freq_{Target}$	Duration
Zynq-7000 AP-vs-AP	DLL	20M	16 MHz	667 MHz	$\sim$ 12 h
STM32MP1 AP-vs-MCU	DLB	10M	15.2 MHz	200 MHz	$\sim$ 9 h
STM32MP1 MCU-vs-AP	DLB	40M	15.2 MHz	650 MHz	$\sim$ 24 h

Table 4.2 Overall delay-line-based power SCA results.

may require additional calibration. For instance, some DLLs can either perform delay calibration continuously or at a set of intervals [7]. Such parameters should be taken into account and calibrated if needed.

The second attack proposed in this chapter required a preliminary work to properly turn the DLB into a custom TDC. Then, two DLB-based power SCAs were conducted on a *STM32MP1* SoC. The AP-vs-MCU AES attack took around 10 million traces for a full key recovery (trace acquisition and CPA took approximately 9 hours) while the MCU-vs-AP AES attack required 40 million traces (24 hours). We can compare these results to the attack reported in Chapter 3 against an OpenSSL AES implementation in an FPGA-based heterogeneous SoC. Using TDC-based sensors we were able to perform a similar attack in only 90,000 traces (FPGA-to-CPU attack). FPGAs indeed offer the possibility to design high resolution and high sampling rate sensors which explain the higher efficiency of their attack. Such a flexibility is obviously not available in ASICs. For instance, even using DMA in our experiments, the maximum sampling rate achieved (16 MHz) was still significantly under the FPGA-based TDC sampling rate (200 MHz). Additionally DLBs also suffer from a poor resolution as evidenced in Figure 4.23 in the appendix. Despite these limitations, we demonstrated that such an attack is still feasible without using FPGAs and within a reasonable time and number of traces.

The presence of DLLs and programmable DLBs is already mandatory in high-end SoC devices and should become even more prevalent in the future with the constant increase of memory bus speeds. At the same time, their voltage sensing capability will be progressively enhanced as they will need to meet higher performances requirements. This should make *SideLine* even easier to conduct and detrimental for hardware security in the future.

4.3.5.2 Hardware & Software Mitigations

This section provides some countermeasure guidelines thwarting *SideLine*:

Adding SCA Countermeasures: A simple way to make the victim process more resilient to power SCAs is the addition of software or hardware SCA countermeasures [149, 147]. As mentioned above, one of the main limitations of *SideLine* comes from the low resolution provided by DLL and DLBs. This forces the attacker to acquire a huge number of traces (several million in our case) and makes it nearly impossible to re-synchronize SCA traces. On the victim side, software randomization could be a good candidate to efficiently de-synchronize computations and hence to increase significantly the attack

difficulty (e.g., adding random delays in T-Table computations for OpenSSL AES). On the monitoring side (delay-line), a straightforward way to mitigate the attack could rely on the addition of phase and frequency jitter to the clock signal used for accessing the delay-line registers.

Preventing Delay-Line Access: Another countermeasure would act at system level by preventing the access to the delay-line registers by unauthorized software entities. Hence, only the OS for instance would have access to this resource. TZ could also be used to place DLLs and DLBs in the secure world and make their use by non-secure world impossible in practice. Locking the access to the DMA module or the hardware performance counters would also represent a significant limitation for the attack setup.

Reducing Delay-Line Sampling Rate: Preventing delay-line access through privilege rights seems insufficient as a malicious attacker or a compromised OS could overpass it (privileges escalation). A hardware way to mitigate the threat would be to limit the DLB access to a lower sampling rate (e.g., 10KHz). This could be simply achieved by limiting the access rate to the register that stores delay-line information. This way, even if the power consumption monitoring would remain feasible, it will highly affect the delay sensor performances. With such a limited sampling rate it would be probably very challenging for an attacker to conduct SCAs on fast encryption algorithms such as AES.

Abandoning Delay-Lines in SoCs: As *SideLine* revealed their potential misuse as power consumption sensors, the delay-line-based components could be removed from SoC devices and instead, be placed directly within the external memory devices. This drastic choice would require the addition of configuration I/Os in external memories to efficiently calibrate the delay-lines but will almost entirely remove the delay-line threat from the SoC die. However, even outside the SoC, the delay-line threat may remain problematic as inter-chip power SCAs have already been shown feasible [121].

4.3.6 Conclusion

Previous works demonstrated that remote power SCAs were feasible using FPGA-based delay sensors and microcontroller ADC-based sensors. In *SideLine* we went further by proving that unsuspected hardware components available in a broad range of high-end SoC devices, can be turned into power consumption measurement units. In this section, we studied two common SoC resources known as DLLs and DLBs and proved their capability to eavesdrop the voltage activity of cryptographic programs running in different processors. Several core-vs-core attack scenarios on application processors and microcontroller units were conducted. For each scenario, we achieved a full key recovery side-channel attack on the publicly available OpenSSL AES implementation. We believe that these findings open a new era for remote power side-channel attacks. *SideLine* has the advantage of being portable on a wide range of devices as it does not requires the presence of specific circuitry (e.g., FPGA). Because *SideLine* feeds upon SoC complexity, we also believe that it represents a major threat for actual high-end SoC security.

More importantly this threat is likely to scale up in line with the constant performance improvements in SoCs and memory devices.

4.4 Additional Results

This section further demonstrates that the delay-line-based attack scenarios are various and can be used for different purpose such as covert channels and Simple Power Analysis (SPA).

4.4.1 Covert Channels between processes

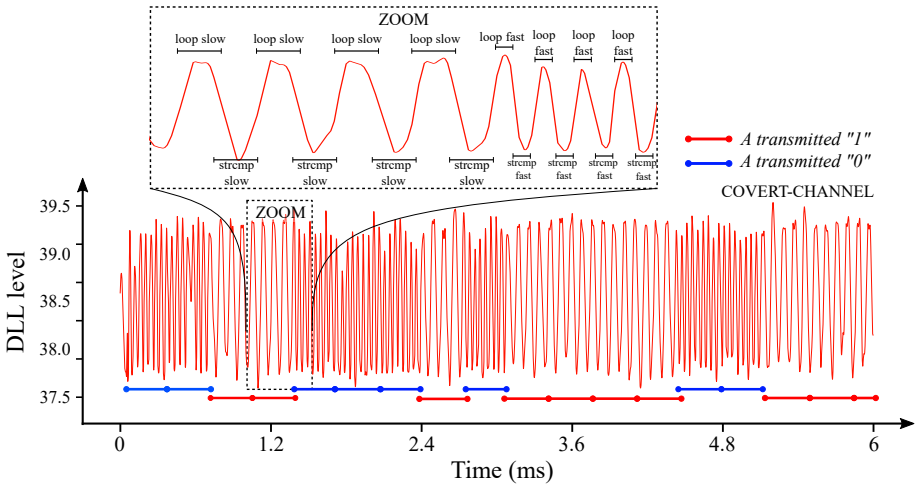


Figure 4.15 Power fluctuation-based covert-channel captured using delay-lines

This paragraph demonstrates how delay-lines could be maliciously used to establish covert channels between applications and SoC components. Covert channels are employed to establish hidden communications between two entities that are not supposed to exchange data. They can be used to extract the information leaked by a software or hardware Trojan or to break the isolation between sandboxed applications.

Here, we aim at monitoring the voltage fluctuations induced by a program using the delay-lines to create a covert-channel between two applications. To that end, we reemploy the `strcmp` proof of concept described in Subsection 4.3.4.2.

Two applications are deployed: a sender and a receiver. The sender runs a simple algorithm that converts a text to be sent into binary data. Then, it modulates its power consumption depending on the data bit to send by alternating between power consuming and idle operations. This creates an amplitude oscillation that can be controlled in frequency. The sender modulates this frequency to transfer the data (frequency modulation transmission). Figure 4.15 illustrates the voltage variations measured by reading the delay-line state on the receiver side. On the red trace, the frequency variations between the transmission of a “0” and “1” are clearly visible. Here, the frequency of a “0” is the double of a “1”. This behavior can be obtained through a simple algorithm implemented on the sender side.

The sender code described in listing 2 implements a for loop that iterates over every `bit` of the `Message` to be sent. For each data bit, it tests its state and enters either a

Algorithm 2 Covert-channel pseudo-algorithm (sender code)

```
Input: Message
for bit in Message do
  i = 0
  if bit == 1 then
    while i < 4 do
      strcmp("aaaaaaaaaaaa","aaaaaaaaaaab")
      for(j=0 ; j < 10000 ; j++){}
      i++
    end while
  else
    while i < 8 do
      strcmp("aaaaaa","aaaaab")
      for(j=0 ; j < 5000 ; j++){}
      i++
    end while
  end if
end for
```

while loop with 8 iterations (the bit state was "0") or a while loop with 4 iterations (the bit state was "1"). To create the amplitude difference, the loop content is made out of an `strcmp` operation that induce voltage drops and empty for loop operations that relax the power consumption (represented in the zoom window in Figure 4.15). The frequency modulation is obtained by modifying the number of increments in the loop and the size of the strings compared within the `strcmp` operation (slow and fast operations in Figure 4.15).

On the receiver side, the application continuously reads the delay-line state and obtain a waveform that can be demodulated to retrieve the data bits. We did not implemented the demodulation algorithm since the signals plotted already testified that the covert channel was working properly. However, we tested various covert-channel bandwidths by decreasing the size of the loops and comparisons used for the modulation. The maximum bandwidth achieved reached 6 Kb/s. This internal covert channel could find an application in the future to leak data from a process to another and even from a SoC to another if the power supply are shared between PCB components [121]. Again, to mitigate this threat, the access to voltage sensing vectors such as delay-lines should remain limited to trusted entities.

4.4.2 Simple Power Analysis on RSA

In this subsection, we evaluate delay-line-based SCA on public key algorithms. Three custom Rivest–Shamir–Adleman (RSA)-2048 decryption algorithm versions were deployed on the bare metal Xilinx *Zynq-7000* target. Based on the WolfSSL RSA crypto-library, we designed a naive *square and multiply* RSA, a *square and multiply always* RSA [28] and a *Montgomery powering ladder* [95, 70] RSA. The main difference between each version lies in the implementation of the modular exponentiation:

- The *square and multiply* naive version uses the basic modular exponentiation algorithm which isn't constant time. A key bit equal to "1" will perform two operations (square + multiply) while a key bit equal to "0" will only perform the square operation. This asymmetry should be visible in the power consumption.
- The *square and multiply always* version uses a modular exponentiation algorithm similar to the "naive" version but adds a dummy multiply operation when the key bit is "0" in order to make it constant-time. This slows down the RSA overall computation but balances the number of instruction computed for each key bit. This should make the SPA more difficult since the RSA will behave more regularly.
- The *Montgomery powering ladder* version is optimized for speed but also constant time since it always requires multiply and squaring independently from the key bit value. In contrary to the *square and multiply always* version, every operation conducted in the Montgomery version is necessary.

Two applications were implemented within the Xilinx *Zynq-7000* CPU cores. The first core contained the attacker app which captured the delay-line state using DMA. The second core contained the victim app which implemented the three RSA versions.

4.4.2.1 SPA on *square and multiply* RSA version

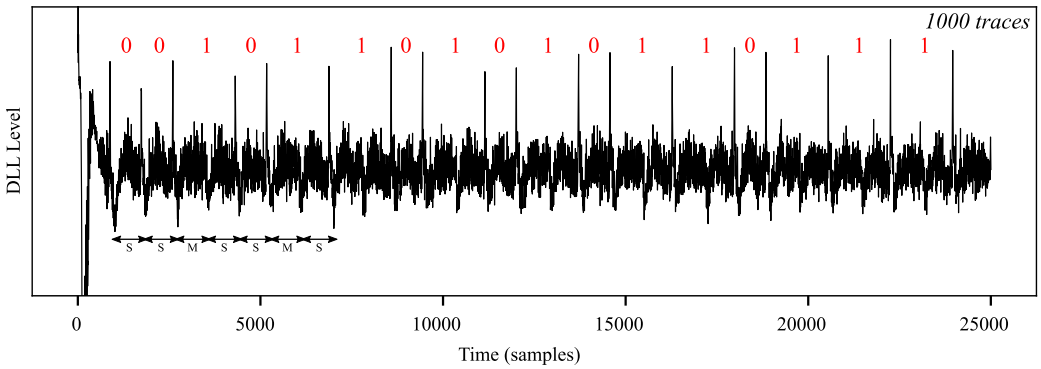


Figure 4.16 *Square and multiply* RSA delay-line-based SPA

Figure 4.16 illustrates the *square and multiply* RSA power consumption captured using the *Zynq-7000* DLLs. The represented waveform only shows the first 17 modular

exponentiation loop iterations over the total 2,048. To collect the entire exponentiation at a 16MS/s sampling rate, 3 million samples need to be captured and stored into DRAM. In Figure 4.16, the square and multiply operation shape can be distinguished. More importantly, a voltage spike arise between each loop increment. The duration for each bit can be precisely measured by taking the distance between each spike. Here, the timing difference between "0" and "1" are clearly visible. The numbers in red in Figure 4.16 represent the private key bits retrieved from the spike distance information. By replicating this method on the entire modular exponentiation function it is possible to obtain the full RSA private key in less than 1,000 traces. The reason why we cannot conduct this attack using a single trace is linked to the DLL precision (illustrated in appendix 4.23 for the AES case). However, by averaging multiple traces, we can progressively reconstruct a detailed signal and obtain a sufficient precision for deducing the key bits.

4.4.2.2 SPA on *square and multiply always* RSA version

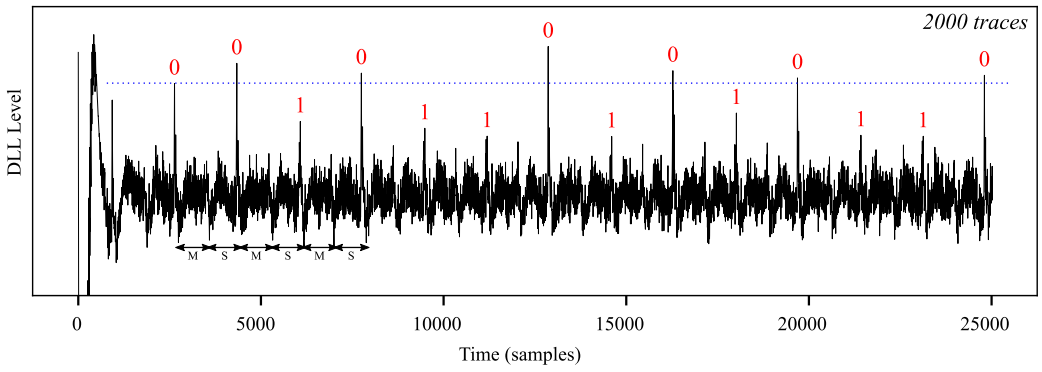


Figure 4.17 *Square and multiply always* RSA delay-line-based SPA

The second RSA version studied is illustrated in Figure 4.17. This time, each key bit was processed with both square and multiply operations independently from its value. In Figure 4.17, the voltage spikes are now equally distant from each other and thus prevent a coarse timing attack from obtaining the key bits. However, the introduction of an else-branch to place the dummy multiplication (in case of a key bit value equal to "0") also has an impact on the amplitude of spikes. It appears that if the key bit value is "0", the spike will be each time bigger than if the bit value is "1". This interesting behavior obtained only by adding the dummy else branch leaks the key by looking at the spikes amplitude instead of the timings. The dotted line in blue helps the reader in differentiating the ones from the zeros. The precision required for this attack demanded more traces. We were able to retrieve the RSA key using delay-lines in around 2,000 trace. In the *Zynq-7000* device and using the wolfssl `sp_arm32` library, the introduction on a dummy else-branch caused the apparition of this amplitude leakage. We did not evaluated this RSA implementation on other devices and thus cannot conclude if this issue is device specific or global. An interested reader

could reproduce the RSA attacks by checking the GitHub repository available here: https://github.com/Remote-HWA/SideLine_Zynq.

4.4.2.3 SPA on *Montgomery Powering Ladder* RSA version

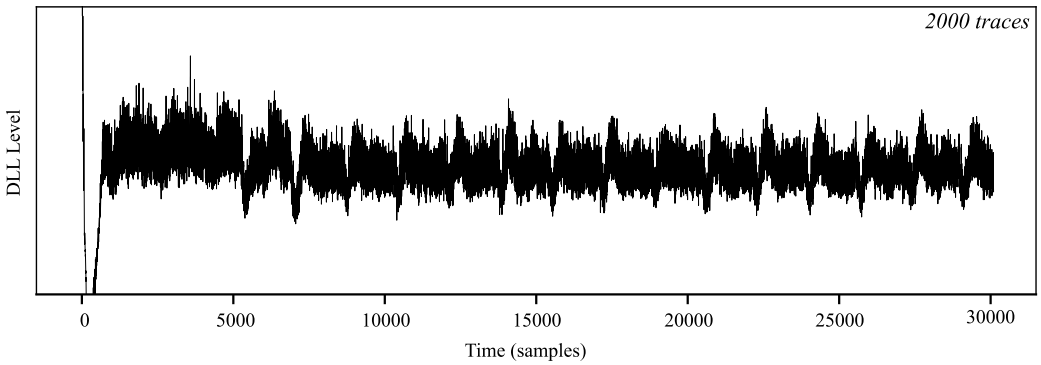


Figure 4.18 *Montgomery powering ladder* RSA delay-line-based SPA

The third RSA version evaluated was based on the Montgomery powering ladder [95, 70] which is intrinsically constant time. Figure 4.18 illustrates the average power consumption observed for this RSA version. Here, the voltage spikes do not appear and cannot be anymore used for amplitude or timing measurements. Even if the shapes observed seem asymmetric, we weren't able to distinguish differences linked to keys bit values using SPA. To go further, we conducted statistical analysis tests to identify key bit classes, but weren't able to obtain enough information since statistical tests require a lot of traces and the lack of precision of the DLL slowed down the process. In this section, we thus restricted our analysis to SPA but more advanced attacks such as horizontal and vertical CPA attacks [26, 11] could be conducted to extract secrets from this RSA implementation. For reasons of time, we let future research works evaluate this aspect of delay-line-based power SCA.

4.5 Conclusion on Delay-Line-based Power Analysis

This chapter was dedicated to the study of delay-line-based power SCA attacks in complex SoC. It demonstrated how an internal component can be identified and turned into a voltage sensor. Then, it explained how these mechanisms can be used along with novel software SCA techniques (DMA sampling, hardware performance counter-based synchronization) to eavesdrop the power leakage of a victim application and steal its secrets.

4.5.1 Results Reminder

The experimental results described in this chapter were published in [53] with co-authors Jean-Max Dutertre, Yannick Teglia and Philippe Loubet Moundi under the name "Side-Line: How Delay-Line (May) Leak Secrets from your SoC".

Responsible Disclosure: We responsibly disclosed our findings to Xilinx on September 22th, 2020 and STMicroelectronics on November 2nd, 2020. Both acknowledged and agreed on the publication of these results. Moreover, this disclosure led to a close collaboration with these companies to find and build efficient countermeasures against *SideLine* and similar attacks. Please keep in mind that *SideLine* has been performed on these two processors for demonstration purposes but that the concept is generic and that any devices embedding delay-lines can be affected.

4.5.1.1 Main Contributions

The works conducted in this chapter led to:

- The discovery of a novel SbSCA attack vector based on delay-lines widely implemented in modern SoC devices.
- A method for efficiently collecting the delay-line state, reducing the effect of OS desynchronization and converting delay values into voltage levels.
- The first SbSCA CPA and SPA attacks conducted on complex SoC systems using internal sensors.

4.5.1.2 Some Numbers

Here we lay out some numbers to remind delay-line-based sensing performances.

Sensor Characteristics:

- Delay-line block based sensors operate at **15.2 MHz** frequency in *STM32MP1* devices.
- DLL-based sensors operate at **16 MHz** frequency in *Zynq-7000* devices.

Attack Results:

- A **delay-locked-loop** can retrieve a software AES key in around **20 million** SCA traces (Xilinx *Zynq-7000* SoC @666MHz).

- A **delay-locked-loop** can retrieve a software naive RSA private key in around **1,000** SCA traces (Xilinx *Zynq-7000* SoC @666MHz).
- A **delay-locked-loop** can be used to establish a covert-channel between two applications. No averaging is required and a 6 KB/s bandwidth can be reached (Xilinx *Zynq-7000* SoC @666MHz).
- A **delay-line block** can retrieve the key of an AES running within the CPU in around **40 million** SCA traces (*STM32MP1* SoC @650MHz).
- A **delay-line block** can retrieve the key of an AES running within the MCU in around **10 million** SCA traces (*STM32MP1* SoC @650MHz).

4.5.2 *SbHA Knowledge*: SbHA Attack Vector Detection Methods

In this section we propose additional methods to identify SbHA vectors within a targeted system. While in this chapter we only relied on data-sheet research, various other techniques can be deployed to detect and use these vectors. By providing these information to the reader, we aim at helping SoC designers and OS developers in identifying these elements and preventing their access to future attackers. Three distinct methods are proposed for detecting SbHA vectors: listing OS user-exposed hardware controls, searching registers in the target documentation and reversing OS device drivers and boot code.

4.5.2.1 Listing OS user-exposed hardware controls

When an OS is implemented within a target, the most straight-forward way to detect a SbHA vector is to study the list of OS command available to the user. Various attacks have been exploiting user-exposed kernel modules that can either provide access to CPU cores power consumption (e.g., `powercap` command in [85]) or enable DVFS regulator parameters modifications (e.g., `CPUfreq` command in [128] and `wrmsr` in [100]). User-exposed kernel modules controlling sensors and actuators are available in almost every OS. However, they provide non-optimal interfaces (slow because using system calls, not optimized for fine-grained attacks) that tend to reduce the attack precision/efficiency (e.g., only a 20 KHz sampling frequency for the `powercap` interface in [85]). A more efficient approach consists in accessing directly the SbHA vector registers using their physical addresses. This can greatly improve the SbSCA sampling rate or the SbFIA glitch precision but also requires a stronger knowledge of the target's hardware. The two detection methods described below use this approach.

4.5.2.2 Searching registers in the target documentation

In *SideLine*, delay-lines were identified by reading reference manuals, data-sheets and application notes from various SoC vendors. This method made it possible to obtain hardware register addresses dedicated to the delay-line configuration and later directly access them from a malicious process. There are tens of processor vendors on the mar-

ket. Among them, a significant part provides public development boards and hardware documentation. These can be directly read by the adversary to identify potential SbHA vectors. The remaining processor vendors only bring portions of the documentation to the public and only share precise details with clients through Non-Disclosure Agreement (NDA) contracts.

Security by obscurity is not a viable solution against SbHA. Mainly because other methods for accessing these hidden vectors were identified. For instance, we observed that a large part of the device studied in *SideLine* embedded IPs brought by specialized vendor. This led us to the conclusion that devices having close properties (same performances, same CPU architecture) often embed equivalent IPs. In the *Zynq-7000* and *STM32MP1* reference manuals it is for instance clearly stated that the memory controllers are IPs brought by Synopsis. If the *Zynq-7000* was an undocumented SoC, we could have found the DLL thanks to our previous works built on the documented *STM32MP1* and vice-versa. Even if the DLL addresses could change, several similarities such as identification registers could have been used to detect the delay-lines within the memory map. Similarities between distinct devices betray their hardware architecture and can be used to retrieve SbHA vectors.

To conclude on data-sheet research, the crucial issue is the adversary ability to identify SbHA vectors within the target's documentation and more importantly their physical addresses. If data-sheets are unavailable the adversary will need to use a software approach to detect known components within the target. The next subsection explains how the attacker could try to reverse kernel drivers to obtain more information on SbHA vectors location and facilitate their access.

4.5.2.3 Reversing OS device drivers and boot code

If the chosen SbHA vector is not software-exposed or cannot be found within the target documentation, an additional identification method based on reverse-engineering can be employed to gain its access. The idea is to study the OS kernel modules source code or binaries to retrieve the SbHA vector physical addresses.

Again, we describe this method using the delay-line example. Because every device using external memories must configure its memory controller to ensure proper communication, there is necessarily an OS kernel module or a function in the OS boot code dedicated to the delay-line calibration. A malicious user could try to access their contents to retrieve the delay-line addresses. For instance several memory controller drivers can be found in the Linux kernel documentation [130] and could be used to locate the delay-lines. If the targeted driver isn't proprietary, this method can be directly applied on the source code. If the driver source code isn't accessible, reverse engineering tools such as *Ghidra* [101] or *Radare2* [108] can be used to retrieve valuable information from the kernel module binaries.

Reading hardware documentation, listing OS commands and reversing OS device

drivers are three powerful methods that could be employed by attackers to gain access to SbHA vectors. Depending on the studied target, the adversary can choose the most convenient one. These three methods involve various computing and electronic skills and further demonstrate that SbHA necessitate both software (for binary reverse-engineering) and hardware (for SbHA vectors identification) expertise.

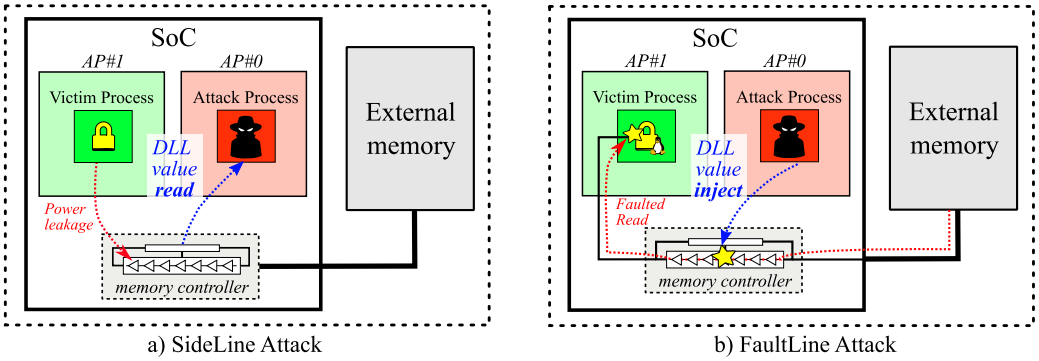


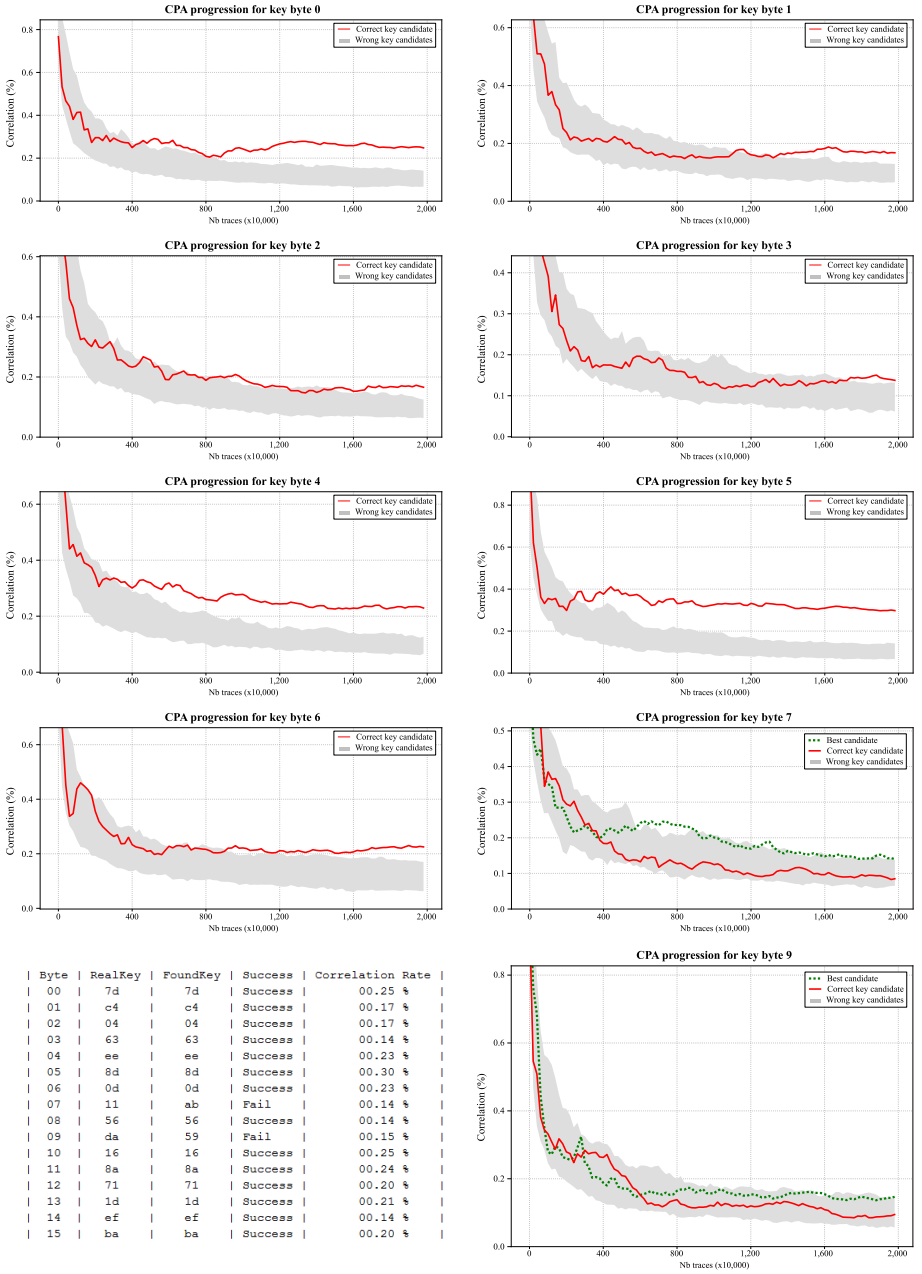
Figure 4.19 From *SideLine* SbSCA to *FaultLine* SbFIA Exploits

4.5.3 From SideLine to FaultLine

Figure 4.19.a illustrates the *SideLine* threat model. A malicious application accesses a delay-line to collect the SCA leakage induced by a concurrently running victim process. The main purpose of the delay-line: delay the clock signal to ensure proper communication with external memories, is not taken into account in *SideLine*. The attack only focuses on the relationship between the delay-line state and the chip's internal voltage. Nonetheless, modifying the delay-line calibration could have disastrous effects on the memory load and store operations launched by the CPU.

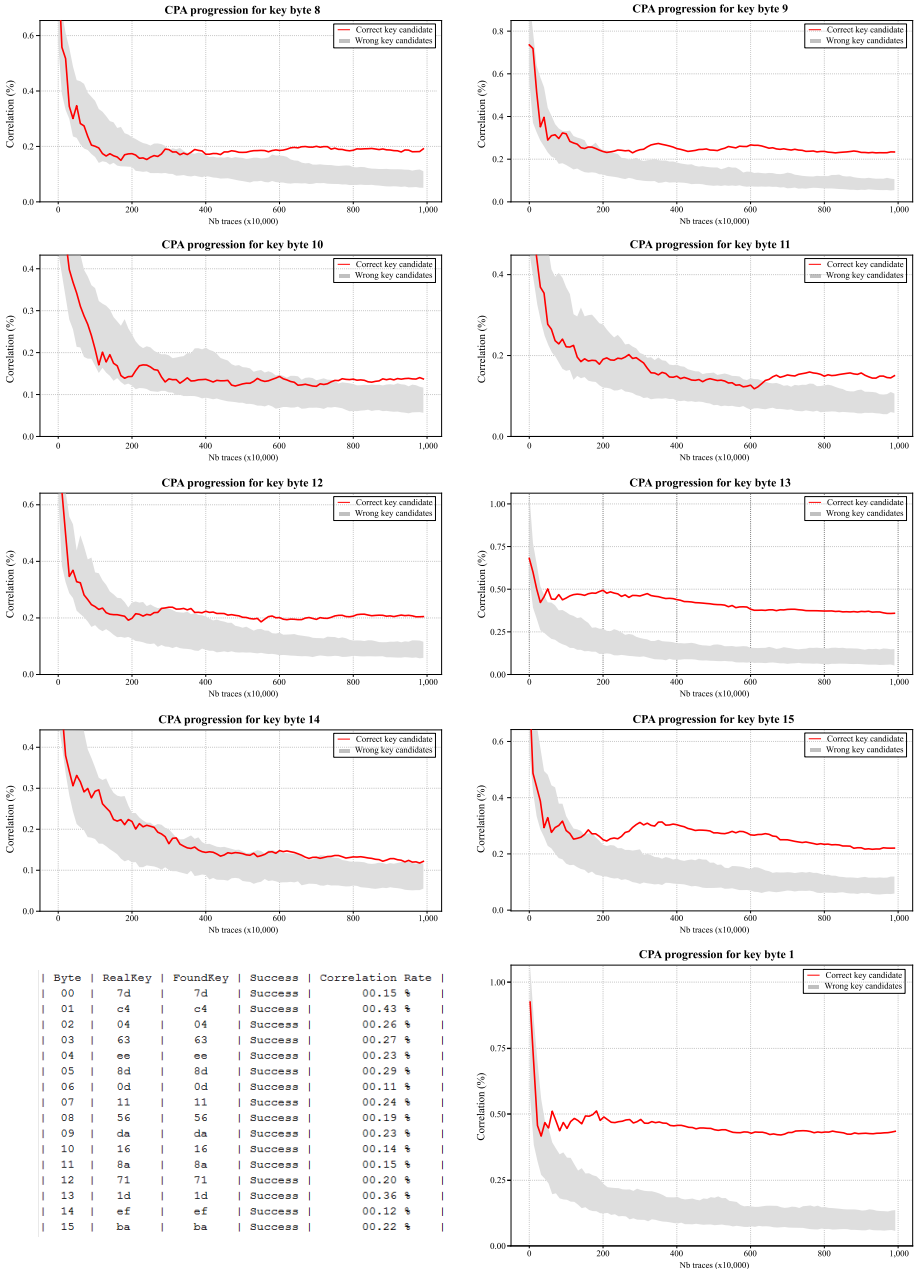
In Chapter 5, we will describe how we turned the delay-lines SbSCA attack vector into a SbFIA attack vector and how it led to the *FaultLine* attacks. As illustrated in Figure 4.19.b, *FaultLine* demonstrates that a malicious application may modify the delay-line state to inject memory access errors in concurrently running applications. This work differs from the previous chapters as it focuses on fault injection. It introduces a novel method for injecting errors from software and, to our knowledge, it is the first of its kind that operates using delay glitches and that targets memory transfers.

4.6 Appendix



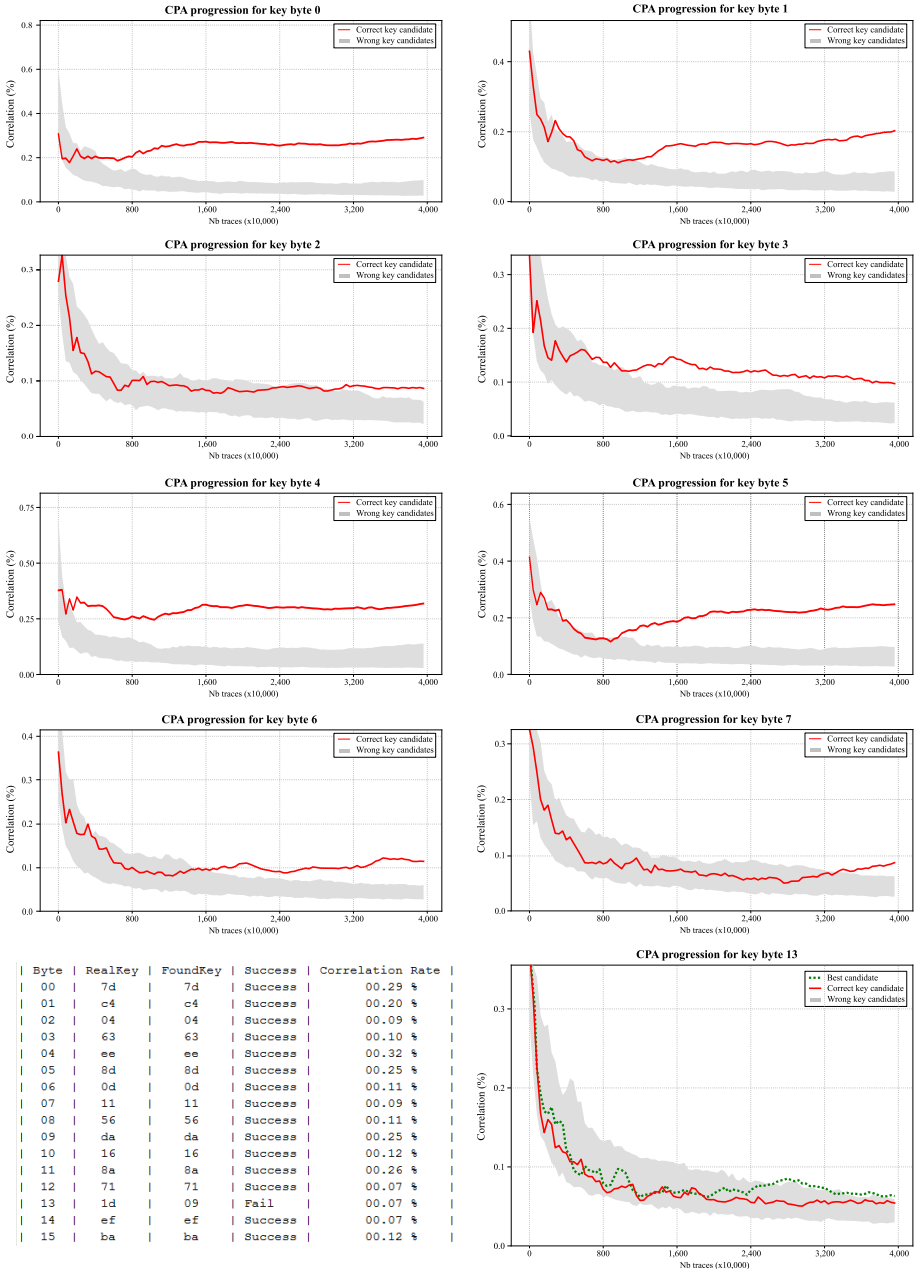
Byte	RealKey	FoundKey	Success	Correlation Rate
00	7d	7d	Success	00.25 %
01	c4	c4	Success	00.17 %
02	04	04	Success	00.17 %
03	63	63	Success	00.14 %
04	ee	ee	Success	00.23 %
05	8d	8d	Success	00.30 %
06	0d	0d	Success	00.23 %
07	11	ab	Fail	00.14 %
08	56	56	Success	00.14 %
09	da	59	Fail	00.15 %
10	16	16	Success	00.25 %
11	8a	8a	Success	00.24 %
12	71	71	Success	00.20 %
13	1d	1d	Success	00.21 %
14	ef	ef	Success	00.14 %
15	ba	ba	Success	00.20 %

Figure 4.20 *Zynq-7000* AP-vs-AP attack scenario - The CPA progression (y-axis) over the number of traces (x-axis) is represented for the first 8 AES key bytes. Bytes 7th and 9th which never emerged from the incorrect key candidates are also represented. These CPA results were obtained over 20 million AES encryptions, the correlation rates are provided in the summary table.



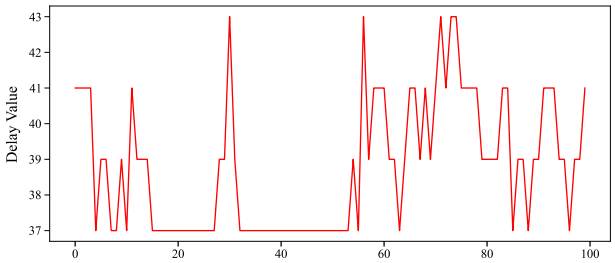
Byte	RealKey	FoundKey	Success	Correlation Rate
00	7d	7d	Success	00.15 %
01	c4	c4	Success	00.43 %
02	04	04	Success	00.26 %
03	63	63	Success	00.27 %
04	ee	ee	Success	00.23 %
05	8d	8d	Success	00.29 %
06	0d	0d	Success	00.11 %
07	11	11	Success	00.24 %
08	56	56	Success	00.19 %
09	da	da	Success	00.23 %
10	16	16	Success	00.14 %
11	8a	8a	Success	00.15 %
12	71	71	Success	00.20 %
13	1d	1d	Success	00.36 %
14	ef	ef	Success	00.12 %
15	ba	ba	Success	00.22 %

Figure 4.21 ***STM32MP1*** **AP-vs-MCU attack scenario** - The CPA progression (y-axis) over the number of traces (x-axis) is represented for the last 8 AES key bytes. The 1st AES key byte is also represented as it provided the best correlation rate. These CPA results were obtained over 10 million AES encryptions, the correlation rates are provided in the summary table.

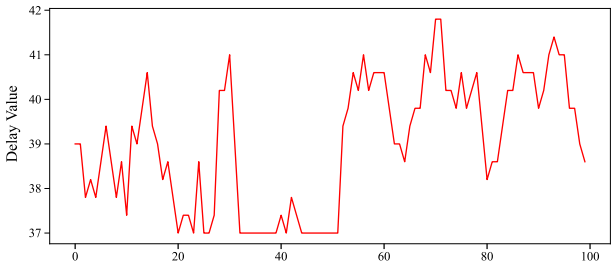


Byte	RealKey	FoundKey	Success	Correlation Rate
00	7d	7d	Success	00.29 %
01	c4	c4	Success	00.20 %
02	04	04	Success	00.09 %
03	63	63	Success	00.10 %
04	ee	ee	Success	00.32 %
05	8d	8d	Success	00.25 %
06	0d	0d	Success	00.11 %
07	11	11	Success	00.09 %
08	56	56	Success	00.11 %
09	da	da	Success	00.25 %
10	16	16	Success	00.12 %
11	8a	8a	Success	00.26 %
12	71	71	Success	00.07 %
13	1d	09	Fail	00.07 %
14	ef	ef	Success	00.07 %
15	ba	ba	Success	00.12 %

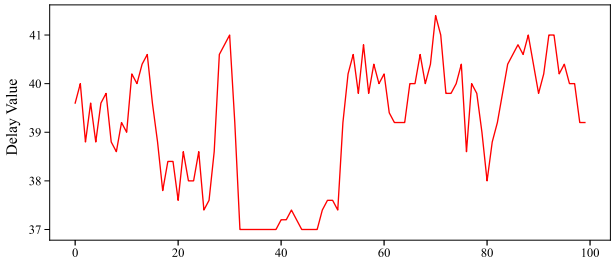
Figure 4.22 *STM32MP1* MCU-vs-AP attack scenario - The CPA progression (y-axis) over the number of traces (x-axis) is represented for the first 8 AES key bytes. Bytes 13th which never emerged from the incorrect key candidates is also represented. These CPA results were obtained over 40 million AES encryptions, the correlation rates are provided in the summary table.



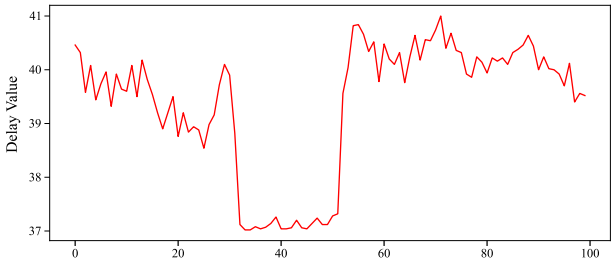
a) Delay value obtained for a **1** AES encryption



b) Averaged delay value obtained for **5** AES encryptions



c) Averaged delay value obtained for **10** AES encryptions



d) Averaged delay value obtained for **100** AES encryptions

Figure 4.23 *STM32MP1* MCU-vs-AP attack scenario: This figure illustrates the DLB resolution limitation when a single AES encryption is acquired (a). This resolution can be virtually increased by averaging a higher number of traces: 5 (b), 10 (c) and 100 (d) traces.

Chapter 5. Software-based Fault Injection on SoC External Memory Transfers

Abstract

In this chapter, we introduce the concept of delay-line-based fault injection. Unlike the previous chapters, this work studies Software-based Fault Injection Attack (SbFIA) instead of Software-based Side-Channel Analysis (SbSCA) and thus describes another scope of application for Software-based Hardware Attacks (SbHAs).

First, we demonstrate that by modifying the delay-line calibration value through a simple register access, a malware may induce faults in memory transfers and jeopardize the security of concurrently running assets. Then, we experimentally evaluate the fault injection on an Operating System (OS)-capable system-on-chip by exposing cryptographic applications to corrupted data and retrieving their secret keys. We finally discuss why delay-line-based fault injection should be systematically considered as a potential threat in modern systems where entities with different privileges share external memories.

Chapter Contents

5 Software-based Fault Injection on SoC External Memory Transfers 129

5.1 Chapter Introduction 130

5.2 Technical Background 131

5.3 FaultLine: Software-based Fault Injection on Memory Transfers 136

5.4 Conclusion on Delay-Line based Fault Injection 149

5.1 Chapter Introduction

In chapter 4, delay-line components implemented in device memory controllers were studied as potential SbHA vectors. We exploited the delay-line relationship with voltage fluctuations in order to perform power Side-Channel Analysis (SCA) attacks. By reading the delay-line state and turning it into leakage information, *SideLine* demonstrated that unsuspected hardware components may be misused by a malware in order to conduct SbSCA attacks.

In this chapter, we introduce a novel fault injection mechanism suitable for SbHA. Similarly to *SideLine*, we focus on programmable delay-lines components widely available in modern systems that use external memories. However, instead of monitoring the delay-line state, we force it to a faulty value. Therefore, we turn the passive side-channel attack vector into an active fault injection medium. Our contributions are described below:

- We reveal that delay-lines components available in a wide range of memory controllers can be turned into memory transfer fault vectors.
- We carry out an extensive characterization of the fault vector. We provide guidelines to control the fault and statistics on the gathered errors.
- On a Xilinx Zynq development board, we evaluate the attack vector in bare-metal mode. We deploy two programs, victim and attacker, and inject faults on the Synchronous Dynamic Random-Access Memory (SDRAM) accesses launched by the victim.
- We then evaluate the fault injection vector while running a Linux-based OS and deploy attacks on Advanced Encryption Standard (AES) encryption and Rivest–Shamir–Adleman (RSA) signature applications.

The remainder of this chapter is organized as follows. In Section 5.2, we provide background information on fault injection attacks and describe the threat model. Then, we present the experimental setup and deploy the attack scenarios in Section 5.3. Section 5.4 concludes the chapter.

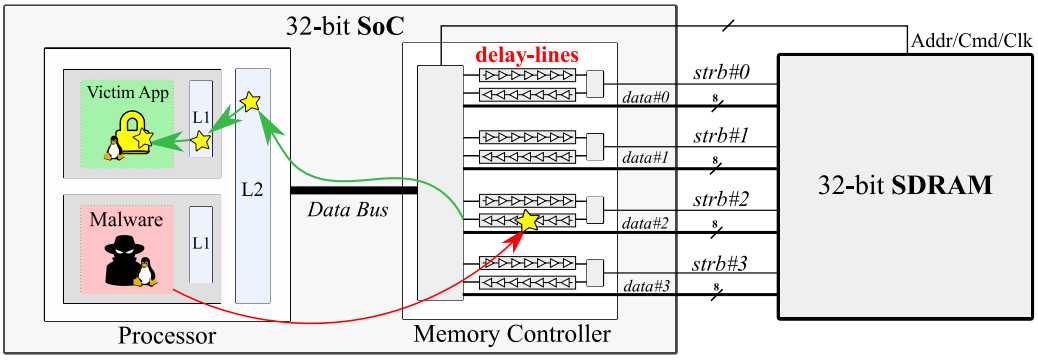


Figure 5.1 Memory transfer organization between a SoC and an external SDRAM memory chip.

5.2 Technical Background

This section covers the technical background related to delay-line-based Fault Injection Attack (FIA). It also describes the persistent fault attack concept later used in this chapter.

5.2.1 Monitoring Memory Transfers

Here we recall the memory transfer principles described in Chapter 4 before introducing the fault concept. Figure 5.1, illustrates the typical components involved in a memory transfer operation between a System-on-Chip (SoC) and an external SDRAM memory. The keystone of this structure is the integrated memory controller. Its main role is to collect and schedule memory access requests. Moreover, it ensures proper timings for the data signals flowing from the SoC to the memory, and vice-versa.

A Double Data Rate (DDR) memory fetches data on both rising and falling-edges of a strobe signal (the strobe is the clock signal for the data lines, each data byte has its own strobe). The high throughput achieved by DDR3+ memories increases the risks of timing errors during a data fetch especially if the data lags behind the strobe signal. The signal lag depends on the length of the PCB tracks which connect the SoC to the external memory. Moreover, it also changes dynamically with voltage and temperature variations. To preserve memory operations from timing errors induced by these lags, recent memory controllers use a dedicated hardware block usually called Physical Layer (PHY) controller. The PHY implements, among others, hardware synchronization mechanisms such as delay-lines (depicted in Figure 5.1) to preserve memory operations from timing errors. The idea is to delay the strobe signal with respect to the data in order to make sure that fetching only occurs when the data is ready.

5.2.2 Faulting Memory Transfers

Because delay-line settings can be accessed and modified during run-time, it may be maliciously used to corrupt memory transfers. Figure 5.1 illustrates a simple scenario in

which a malware (in red) and a victim application (in green) run concurrently on a SoC processor cores. Both apps occasionally generate memory accesses that are handled by the integrated memory controller. In this particular example, the malware aims at corrupting the data loaded by the victim app. To that end, it modifies the delay-line calibration value for a short period of time (red arrow). The calibration chosen is a critical value that is known to induce faults during reads. With any luck, the victim app is simultaneously performing a load operation which therefore results in a corrupted data read. The fault is then loaded into the cache memory (green arrow) and remains persistent until the faulted data is evicted. Depending on the loaded data, the fault injection may jeopardize the victim app and its security. Several attack scenarios presented in Section 5.3.2 and 5.3.3 arise from this fault model.

5.2.3 Threat Model

Modern systems are more and more heterogeneous. They embed, processors, microcontrollers, Field-Programmable Gate Arrays (FPGAs), secure elements, etc. All these assets may use common external memories and thus share the memory controller in order to access data. While logical isolation theoretically prevent processes from accessing the others, they cannot detect tampered data transfers induced by a malware.

Here, we further specify the threat model by illustrating it with the example of a SoC embedding both a user and a secure processor. The 1st one or "user processor" implements a rich OS and contains the user data. The 2nd one or "secure processor" performs security related computations: OS verification, login, encryption/decryption. The sensitive data belonging to the secure processor such as cryptographic keys and secure applications are protected by a Trusted Execution Environment (TEE) based for instance on TrustZone (TZ) [8] and/or a hardware security module. Let's suppose that the user processor is compromised by a hardware level malware or a malicious OS which aims at extracting a protected key. In an attempt to bypass the security isolation, the user processor leverages its access to delay-lines to corrupt memory transfers handled by the secure processor. A fault injected inside a crypto-algorithm can then be exploited to expose its cryptographic key. To that end, the malicious processor leverages traditional fault analysis methods such as Differential Fault Analysis (DFA) [110] or Persistent Fault Attack (PFA) [146].

5.2.4 Persistent Fault Analysis on AES

In this Subsection, we aim at describing the concept behind PFA as this attack will be used in this chapter to extract AES keys from victim processes. In contrary to the DFA techniques used in this chapter that were introduced by Piret & Quisquatter in 2003 [110], the PFA attack technique published in TCHES 2018 by Zhang et al. [145] is recent and thus lesser known. Therefore, it seems interesting to describe its concept.

As its name suggests, PFA takes advantage of a fault that persists over successive encryptions. Such kind of fault is easy to obtain in practice using hardware injection tools such as lasers, ElectroMagnetic (EM) and power glitch injectors. For instance a laser may permanently modify the value of a memory register. By faulting a memory transfer, an EM or power glitch may generate a corrupted read operation that will be stored persistently within the cache memory until the next eviction or in SDRAM until reset. In complex systems using cache memories, persistent faults are more likely to occur as every data or instruction may be stored temporarily in cache. For this reason, PFA stands as a good candidate for extracting secrets from modern SoC devices. The PFA attack on AES can be split into 5 main steps. Here we describe each step chronologically:

1. **Inject a fault**

   The PFA model presented in [145] requires a fault injected within one element of the AES Sbox table. Consider the example given in equation 5.1, it depicts a suitable fault injected within the first Sbox element. Here, its value was modified from 0x63 to 0x41.

$$Sbox[0] \rightarrow Sbox'[0] = 0x63 \rightarrow 0x41 \tag{5.1}$$

   The PFA attack only works if a single Sbox element is modified. Additionally to this requirement, the fault must persists over successive encryptions in order to generate enough faulty ciphertexts for the statistical analysis (step 3).

2. **Collect thousands of faulted ciphertexts**

   According to [146], the collected batch must contain at least 1,641 faulted ciphertexts for enabling key retrieval. The access to plaintexts is not required.

3. **Compute the faulty ciphertext distribution**

   Once all the ciphertexts have been extracted (at least 1,641), the attacker still doesn't now if the fault was successfully injected within a single Sbox element. However, and this is the core concept behind PFA, he can get this information by evaluating the distribution of ciphertexts.

   Since one Sbox element disappeared, the Sbox table is not anymore bijective. One Sbox element does not anymore exist in the table (0x63 in example 5.1) and another one now appears twice in the table (0x41 in example 5.1). Equation 5.2 illustrates how this Sbox table modification impacts the probability distribution Pr of the ciphertext values obtained.

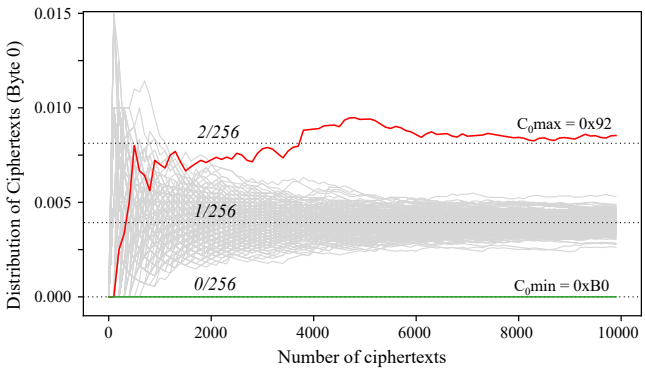


Figure 5.2 Exemplary AES ciphertext byte 0 distribution with a fault injected on one Sbox element

$$\begin{aligned} Pr = 0/256 \quad & c_j^{max} = Sbox'[i] \oplus K_j \\ Pr = 2/256 \quad & c_j^{min} = Sbox[i] \oplus K_j \\ Pr = 1/256 \quad & otherwise \end{aligned} \tag{5.2}$$

j represents the j^{th} byte of the ciphertext (16 bytes per ciphertext). i represents the unknown index of the modified Sbox element. Since the last AES round does not involve a MixColumn operation, the collected ciphertexts values are only defined by the output of the Sbox table XORed with the last round key (as depicted in equation 5.2). Thus, in case of a successful injection, one ciphertext byte value will never occur c_j^{min} (0/256) and one ciphertext byte value will appear twice as much than the others c_j^{max} (2/256). Therefore the ciphertext distribution won't be anymore uniform. This is exemplified for a ciphertext byte in Figure 5.2. For each ciphertext byte j, c_j^{min} and c_j^{max} take a different value.

By computing the ciphertext byte distribution, the attacker immediately knows if a single Sbox element was faulted. If it is not the case, the attacker returns to step 1 to retry the injection. At the end of step 3, the attacker knows the 16 c_j^{min} and c_j^{max} values corresponding to each ciphertext byte.

4. **Retrieve the Sbox index**

   Once step 3 has been fulfilled, the attack has succeeded. However, the adversary needs to conduct a statistical analysis to extract the key. Indeed, c_j^{max} and c_j^{min} are known but $Sbox[i]$ or $Sbox'[i]$ should be defined to obtain key.

   In order to retrieve the last round AES key K10, the attacker must find the Sbox value missing in the table $Sbox[i]$. That is: $k_j = c_j^{min} \oplus Sbox[i]$. To that end, he computes 256 candidates for K10 according to the 256 possible values for the missing Sbox value $Sbox[i]$. The adversary then obtains 256 last round key candidates K10 but only one is the correct key.

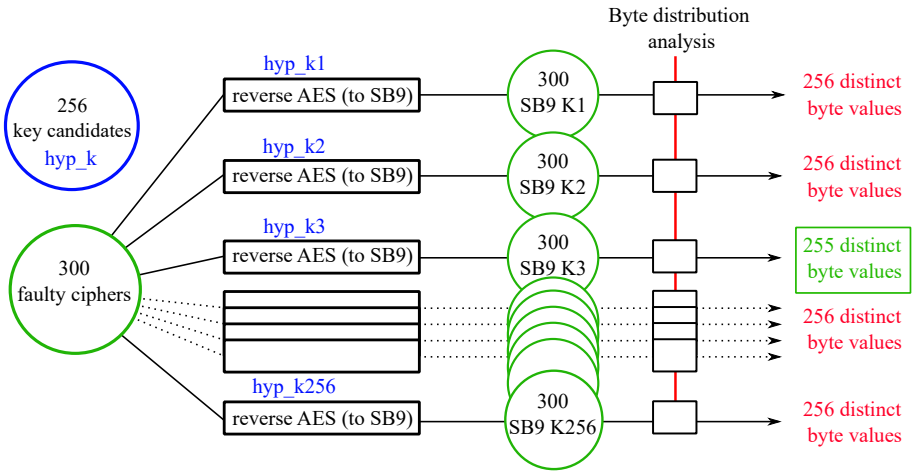


Figure 5.3 Method for extracting the right PFA key candidate (without the use of a reference plaintext/ciphertext pair)

5. **Deduce the key**

   If the attacker has access to a reference plaintext/ciphertext pair (non-faulted), he can proceed with the encryption of the plaintext with each key candidate until a ciphertext that matches the reference is found. This brute-force method works but the adversary hasn't necessarily the possibility to access such a reference pair in practice.

   Another method that does not require this pair is proposed in [146] and is illustrated in figure 5.3. Using the 256 key candidates computed in step 4, the faulted ciphertext batch (a minimum of 300 ciphertexts is required this time [146]) can be decrypted to the output of the SubBytes operation in the penultimate AES round ($SB9$). All the SB9 batches generated with a wrong key candidate will produce 256 different $SB9$ output values. And, because the persistent fault can cross multiple rounds, only the right key candidate will provide 255 $SB9$ output values for the entire batch. Based on this missing value the right key candidate is found.

PFA is a powerful attack technique that applies on block cipher cryptosystems. It was extended to the PRESENT [14] algorithm in [146] and to masked AES implementations in [107]. In Section 5.3, we use this method to extract keys from software AES processes.

5.3 FaultLine: Software-based Fault Injection on Memory Transfers

In this section, we implement the *FaultLine* attack concept. On a Xilinx Zynq development board, we evaluate the delay-line-based SbFIA vector by injecting faults on SDRAM accesses.

5.3.1 Experimental Setup and Fault Parameters

As *SideLine*, *FaultLine* is generic as delay-lines are implemented in a wide-range of modern devices. As other fault injection vectors it is configurable and should be characterized to maximize the fault rate/success.

5.3.1.1 Experimental Setup

The target adopted to demonstrate our fault mechanism is the ZYBO development board from Digilent `PB200-351 REV B` [65]. This board embeds a Xilinx Zynq SoC `XC7-Z010-1CLG400C` [141] and two 16-bit DDR3 memory chips of 512 MB each.

The Zynq processor contains two ARM Cortex-A9 cores cadenced at 666 MHz and an Artix-7 FPGA (not used in this work). Figure 5.1 illustrates the simplified view of the Zynq memory organization and interfaces. Each processor core contains independent 32 KB level 1 instruction and data caches. It also integrates a 512 KB level 2 cache which is shared between the two processors.

Two software setups were considered in this work:

a) ***bare-metal*** *setup:* These experiments were built using Xilinx Vitis version 2020.2. We leveraged the possibility to load one executable per Computer Processing Unit (CPU) core to build dual-core attacks. The bare-metal configuration requires a hardware platform specification file for the ZYBO and the board support package project generated in Vitis. Note that the used Vitis and Vivado versions should not impact the attack results. The algorithm attacked in this scenario is the TinyAES available on GitHub [78]. Bare-metal attacks are described in Subsection 5.3.2.

b) ***Linux-based*** *setup:* We implemented the Linaro Debian Linux OS version *linaro-jessie-developer-20161117-32* on the Zybo board. Our Linux-based attacks were not constrained, we did not perform any kernel modification. We did not limit the execution of background processes or kernel processes. In this scenario we targeted two algorithms: the TinyAES and the OpenSSL crypto-library [106]. Linux-based attacks are described in Subsection 5.3.3.

All the documentation about delay-lines was found in the Xilinx Zynq 7000 technical reference manual [141] chapter *DDR memory controller*, Subsection *PHY controller*. We provide all the bare-metal and linux-based attack codes in the following GitHub repository: `https://github.com/LAbbbs/FaultLine`.

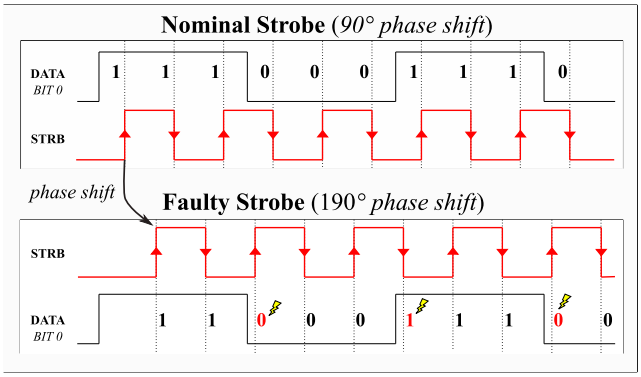


Figure 5.4 Nominal vs faulty strobe phase-shifts

5.3.1.2 Deeper View of the Fault Mechanism

When a read operation is initiated by the memory controller, the SDRAM first collects the data in the proper bank and then outputs the data signals and the strobe signals edge-aligned. These signals then travel through the PCB tracks and reach the SoC and its memory controller. Inside the SoC, delay-lines are used to delay the strobe signal with respect to the data.

The top part of Figure 5.4 illustrates the correct sampling of consecutive reads. Here, the black signal represents the data line 0 out of the 32 total data lines. The strobe signal depicted in red is used as a reference clock to fetch the data. In nominal operation, a 90° phase shift is added to delay the strobe signal propagation. This ensures that the data bit is ready and stable before its capture by the strobe edge. The bottom part of Figure 5.4 illustrates a different delay-line calibration for the same read sequence. This time the strobe phase-shift was increased to 190° while the data phase remained stable. This resulted in read errors in the fetching operation as the strobe delay was too high.

The delay-line calibration is controlled from software and the fault duration can be finely tuned to affect a limited number of read or write operations.

5.3.1.3 Shaping the Glitch

The typical faults obtained using this method are bit flips. Their apparition rate, the number of bits flipped and the number of faulted read and write accesses can be precisely controlled using software parameters. As for power or electromagnetic glitches, delay-line-based fault injection can be finely calibrated to maximize the fault success and limit the number of mutes (processor crashes caused by invalid instructions and segmentation faults). In addition to usual parameters such as glitch strength and width, delay-line-based fault injection bring additional controls on fault direction and faulted-byte. The following list introduces five parameters that need to be tuned using software to inject precise faults in memory transfers. The pseudo-code given in Listing 5.1 illustrates how these parameters are controlled from the malware. We comment them in order of appearance:

Listing 5.1: Fault Injection Pseudo-Code (bare-metal)

```
1 // register addr depends on direction and byte
2 addr = get_delayline_addr(direction,byte);
3 for(i = 0 ; i < init_delay ; i++){} // delay
4 writereg(addr,stress);  // start glitch
5 for(i = 0 ; i < width ; i++){} // glitch width
6 writereg(addr,nominal); // stop glitch
```

1. `direction`: Because there is necessarily a separate delay-line for read and for write operations, the attacker can choose the direction of the fault. This can be controlled by changing the accessed delay-line register (line 2 in script 5.1). On the Zynq processor, we induce a write fault by modifying the `phy_wr_dqs_cfg0-3` registers and we induce a read fault by modifying the `phy_rd_dqs_cfg0-3` registers.

2. `byte`: The fault can be even more targeted as there is a distinct strobe per memory data byte. That is, 4 strobes for 4 bytes. On the Zynq processor, 4 separate registers can be modified to inject a fault on the 4 distinct data bytes. The byte choice parameter added to the fault direction parameter provide a fine and powerful control to the attacker as he can focus on a specific read or write data byte over the 4 existing ones.

3. `init_delay`: This parameter (line 3) acts as a temporization medium prior to the glitch injection. It can be programmed to perform temporal fault injection mappings. In bare-metal mode, we finely tune it using `for` loops. These are easily programmable and are quite accurate (~1 increment per clock cycle).

4. `stress`: This parameter defines the delay-line calibration value, that is, the phase-shift applied on the strobe signal. To generate read or write faults, the malware modifies the stress in order to induce a faulty phase-shift (line 4). This unstable phase-shift can be reached by increasing or decreasing the stress until a first faulted memory transfer occurs. If the stress is too distant from the nominal operation the risk of obtaining mutes soars. Please note that the optimal delay-line calibration value may slightly change from a board to another due to process variations.

5. `width`: The width parameter determines the glitch duration, that is, the time during which the `stress` will be applied on the delay-line. It is controlled by inserting a programmable delay between the glitch injection (line 4) and the glitch relaxation (line 6). In bare-metal this delay is also controlled using `for` loops (line 5). We evaluate its impact in Subsection 5.3.2.

5.3.2 *FaultLine* on a Bare-Metal Device

This section describes fault evaluation and attacks conducted using two parallel programs running within the Zynq processor cores.

5.3.2.1 Characterizing the Injected Faults

The delay-line calibration registers can be set and forced to values ranging from 0 (the shortest delay) to 512 (the longest delay). Within this range, we aim at distinguishing the proper delay calibrations from the faulty ones. To that end, we deployed two C programs running on separate CPU cores. The memory caches L1 and L2 were disabled using the Xilinx API `Xil_DCacheDisable()` to ensure that each load or store operation would result in an SDRAM memory access. The injection program runs in CPU#0, It implements a looped version of Listing 5.1. The victim program runs in CPU#1, it implements the pseudo-code given in Listing 5.2.

Listing 5.2: Victim memory transfers pseudo-code

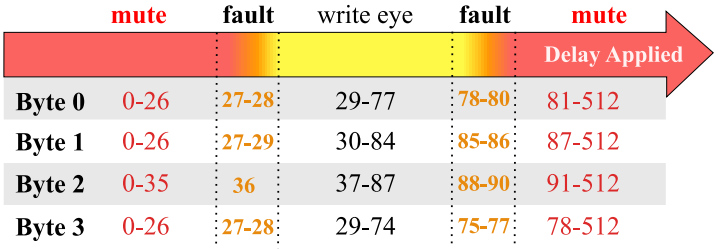
```
1 int ref_array[nAttack]; //nAttack = 1,000,000
2 int store_array[nAttack];
3 DisableCacheL1_L2();  // force SDRAM access
4 FillArray(ref_array,0x00000000,0xFFFFFFFF);
5 // The fault is injected within this loop
6 for(int i = 0 ; i < nAttack ; i++)
7 store_array[i] = ref_array[i];
```

A one million elements `test_array` (line 1) is filled with two successive values: `0x00000000` on even indexes and `0xFFFFFFFF` on odd indexes (line 4). Then a loop copies the `test_array` content within the `store_array` (line 6-7). Thanks to the adopted repetitive pattern, a continuous sequence of zeros and ones will travel through each data line when the array is stored or load. We chose this pattern as a higher number of data line state transitions should maximize the fault rate.

Fault injections occur during the array copy operation. If the fault direction is *read*, it will affect the `test_array` loading from SDRAM. If the fault direction is *write*, it will affect the export of the `store_array` inside the SDRAM. Based on this setup we deployed several tests:

Glitch stress sweep test: For each delay-line calibration values available we performed one million writes to SDRAM. Figure 5.5.a highlights the obtained results from the smallest to the greatest phase-shift applied. Three different colors corresponding to three different behaviors can be distinguished. The yellow one at the center represents delay-line calibration values which gave correct writes (the write-eye) `store_array = ref_array`. On the edges, the red one represents the delay values which induced mutes (communication with the SoC is lost until reset). In this OS-free configuration, mutes are essentially caused by exceptions related to corrupted instructions or invalid memory accesses. Finally, the values displayed in orange are the one that generated usable faults: `store_array != ref_array`. Only few values are suitable for fault injection. These values correspond to the boundary between proper and faulty operations. That is, a maximum negative phase-shift (orange left) or a maximum positive phase-shift (orange right) between the strobe and the data. At the operating boundary, the error rate highly depends on PVT variations that make the strobe signal phase unstable. This insta-

a) Glitch stress sweep test results (write)



Byte 0	0-26	27-28	29-77	78-80	81-512
Byte 1	0-26	27-29	30-84	85-86	87-512
Byte 2	0-35	36	37-87	88-90	91-512
Byte 3	0-26	27-28	29-74	75-77	78-512

b) Glitch width sweep test results (read)

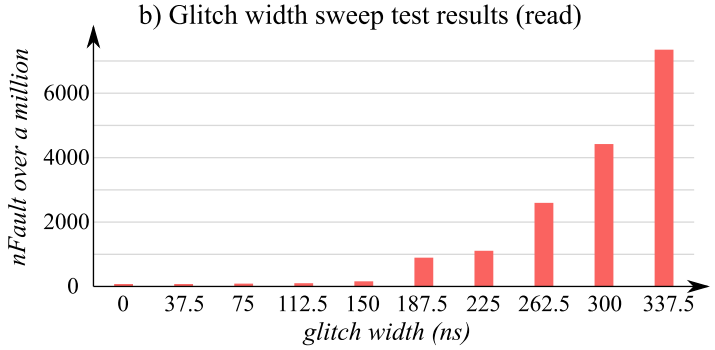


Figure 5.5 `stress` and `width` characterization

bility generates both proper and faulty fetches. To avoid mutes, we try to limit the fault rate to less than 1% by finely tuning the glitch parameters. Under this rate, the attack is almost mute free.

Glitch width sweep test: A linear increase of the glitch width does not necessarily imply a linear increase of the fault number. Figure 5.5.b depicts this behavior obtained on read tests. The fault rate remains quite stable (around 100 over a million) until the glitch width reaches 150 ns. Then it increases much faster between 150 ns and 337.5 ns. Higher width values generated mutes during our experiments. From these results, we can deduce that there is a minimal glitch `width` inherent to the hardware. This explains why the fault rate does not depends on the glitch width applied when it is smaller than 150 ns. Once reached and after 225 ns, the fault rate increase becomes roughly linear. In our following experiments, we tuned the glitch `width` according to the glitch `stress` applied. Proper `stress` and `width` pairs were established using sweep tests.

5.3.2.2 Differential Fault Analysis Attack on AES

The first attack application proposed in this chapter consists in conducting a core-vs-core Piret & Quisquater (P&Q) DFA attack [110]. The attacker and victim programs run in separate CPUs. The victim CPU#1 computes Tiny AES encryptions on attacker CPU#0 request. The programs are loaded through JTAG inside the SDRAM at addresses 0x5000000 for the attacker and 0x6000000 for the victim. To facilitate the DFA Attack, we disabled the data caches L1 and L2. This relaxes the need to perform a cache flush before each fault injection as the Sbox table would have been automatically loaded into cache memory.

```
[victim] pt:  8620914039f68c7c5a33509aecda185a
[victim] ct:  00000000000000000000000000000000
[attacker] byte 0, stress:  1, delay:  1620
[victim] ct:  0000A800008E0000330000000000000090
[victim] ct:  0000490000 3A0000820000000000000066
[attacker] byte 1, stress:  2, delay:  1640
[victim] ct:  2A000000000000D1000045000094 0000
[victim] ct:  A4000000000000820000510000 9F0000
[attacker] byte 2, stress:  56, delay:  1640
[victim] ct:  B4000000000000720000390000 5A0000
[attacker] byte 3, stress:  52, delay:  1620
[victim] ct:  0000008900005600004500 00AA000000
[victim] ct:  0000001000 00FD0000450000 1C000000
```

Table 5.1 Variations in faulty ciphertexts obtained depending on the delay-line byte choice (read direction)

Here is how the attacker proceeds: first, an encryption from CPU#1 is requested. Shortly after, it generates a delay-line glitch according to the mechanism given in Listing 5.1. We chose to conduct a simple DFA attack in AES round 9 as described in [110]. To that end, the glitch must be injected between the penultimate and ultimate *MixColumns* transformations. A fault success can be easily detected as exactly 4 bytes of the faulted ciphertext will differ from those of the correct one.

To find the proper injection timings, our malware gradually increased the `init_delay` with steps of 150 ns. As expected, the first faults injected within the AES computation modified all the ciphertext bytes. Once the injection timing reached the penultimate *MixColumns* we started obtaining 4-byte ciphertext faults. Table 5.1 shows several faulty ciphertexts collected with faults injected during the 9th AES round. For visualization ease, we chose a constant plaintext `pt` which generates an all-zero ciphertext `ct` for a nominal AES encryption. We also used the `byte` parameter as an advantage to inject faults in different AES columns. Indeed, the P&Q attack requires at least 2 faults in each AES column to succeed. The control of the targeted delay-line `byte` allowed us to choose the column under attack (faulty positions in red). This additional control maximizes the chances to obtain new groups of faulty ciphertexts and thus increases the attack speed. Based on our experimental results, it took 100,000 injections (2.4 minutes) and around 500 AES faults in average to retrieve the full AES secret key. Among these 500 faults, 8 only were required for DFA the others were duplicates or faults injected within the same column.

5.3.2.3 Persistent Fault Analysis Attack on AES

The P&Q attack presented previously requires memory caches disabling and thus makes the attack hardly feasible in practice. This is why we introduce a second proof-of-concept of delay-line-based fault injection which relaxes the attacker prerequisites: the PFA [146]. As described in Subsection 5.2.4, the PFA attack leverages the processor capability to temporarily store data and instructions into memory caches instead of performing time-consuming SDRAM memory accesses every time a data is read. If a fault is injected during a memory access from the SDRAM, it will remain persistent in the cache until the data is evicted. It has been experimentally demonstrated in [146] that this effect can be

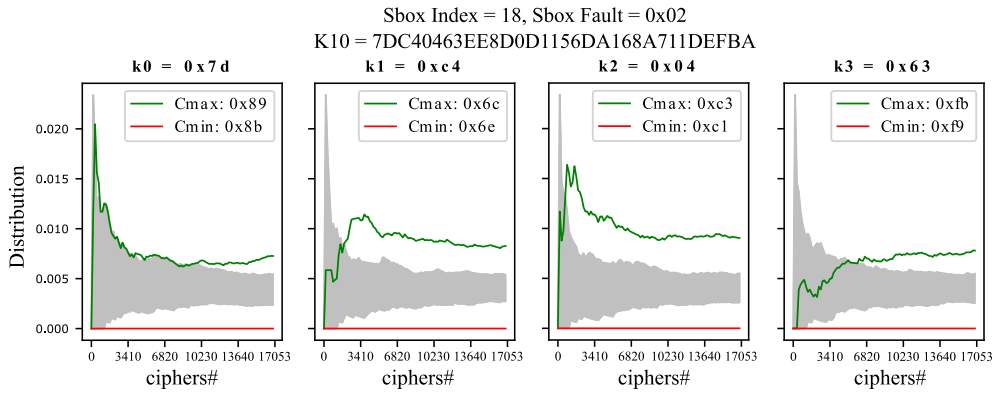


Figure 5.6 Progression of the PFA distribution over 17,053 faulty ciphertexts. The `Cmin` distribution (in red) results from the missing Sbox. The `Cmax` distribution (in green) results from the duplicated Sbox value.

maliciously employed to break symmetric crypto-algorithms.

We deployed PFA on our experimental setup. Again, the CPU#1 computes Tiny AES encryptions on CPU#0 requests. However, this time, both L1 and L2 memory caches are enabled. Because the data loaded into the cache memory remains cached, a fault injected during the first Sbox loading from SDRAM will persist over time. Therefore, on AES, by faulting a single Sbox element, we can expect to continuously obtain faulty ciphertexts. This requires a single fault injection, success can be detected through PFA distribution analysis described in section 5.2.4 and more importantly the attack doesn't require fine synchronization. Here is how the attacker may proceed:

1. Flushing: The attacker flushes the cache memory to remove all the Sbox indexes potentially stored during previous encryptions. The Zynq board support package comes with several APIs to flush the data and instruction caches. Here we used the `Xil_DCacheFlushRange()` function to flush the AES variables.

2. Collecting: The attacker then asks for 20,000 AES encryptions to the victim process (without cache flush). If a fault is detected, the attacker saves the encryptions for post-treatment, else he returns to the first step.

3. Analyzing: Using the collected ciphertexts, the attacker computes the Sbox candidates according to the method described in [146]. If a single Sbox fault was injected he can retrieve the full AES key using less than 2,000 faulty ciphertexts.

In practice, it took less than 10 seconds and around 1,000 glitches to obtain a suitable fault. Based on this faulted Sbox, we computed and collected 17,053 faulty ciphertexts. An illustration of the faulty ciphertext distribution obtained for the first four AES key bytes is provided in Figure 5.6. The c_j^{min} apparition rate (in red) remains zero because its associated Sbox output was removed by the fault. c_j^{min} emerges from the global distribution after around 1,200 ciphertexts. With this information, we can retrieve the missing Sbox index and value. Then we use the following equation $c_j^{min} = Sbox[i] \oplus K_j$ to retrieve the AES key bytes K_j.

Listing 5.3: Linux OS Attack Script

```
sync; echo 3 > /proc/sys/vm/drop_caches
./malware -dly -val -width -dir -byte &
./encrypt -nEnc &
```

5.3.2.4 Conclusion on Bare-Metal Results

In this section, we demonstrated that a simple SoC register modification can lead to a complete disclosure of cryptographic secrets. We demonstrated that DFA and PFA attacks are feasible on a Zynq device using delay-line-based fault injection. The fact that memory caches are enabled doesn't mitigate the attack as long as the attacker finds a method to flush them. Delay-line-based fault injection also provides additional fault parameters such as direction and byte choice. These were used to facilitate both the DFA and PFA attacks.

5.3.3 *FaultLine* on a Device Running Linux

The bare-metal setup presented in Subsection 5.3.2 is convenient for fault characterization as it relaxes the attack complexity. With a Linux Debian OS now deployed on the Cortex-A9 processors, we aim at generalizing, comparing and demonstrating that our attack medium remains functional in real-life complex OS scenarios.

5.3.3.1 Attack Setup

Each attack scenario presented in this section involves two Linux applications: a `malware` and a victim `encrypt` app. The following paragraphs describe the attack scenario.

Attack script: Listing 5.3 illustrates the three commands required to inject a fault. First, the adversary flushes the entire memory cache to make sure that the victim data will be loaded from SDRAM. We used the system call `drop_caches` to flush the data stored in memory caches. Dropping all the memory caches is time-consuming and drastically reduces the fault injection rate. However, to our knowledge, this is the only flushing method available on this platform that do not involve any kernel modification. Then, the two following script lines successively launch the `malware` and the victim `encrypt` apps. The presence of the & symbol at the end of each system call enables concurrent application running. This ensures that the malware and the encryption apps will run simultaneously.

Malware App: In Listing 5.3 line 2, the malware app takes the glitch parameters as arguments. When launched, it first maps the delay-line physical addresses using the `mmap()` privileged system call. Then, it injects the glitch with respect to the pseudo-code given in Listing 5.1 and exits. The `init_delay` mechanism previously implemented using `for` loops was improved with the use of `clock_gettime()`. This system call

directly accesses the hardware performance counters and is therefore less affected by temporal jitter caused by OS interruptions. We leveraged this stable delay medium to perform fine temporal mappings. That is, we gradually delayed the fault injection with respect to the encryption app execution until a fault occurred.

External Computer: An external computer (not required but used for experimentation ease) automates the attack and resets the Zybo board on mutes. The Python Paramiko package is used to script the SSH communication between the computer and the Zybo. The bash script 5.3 is continuously relaunched with new arguments. Once the injection campaign is completed, a post-attack analysis is conducted on the computer side to retrieve the successful errors.

5.3.3.2 Simple Data Byte Corruption

We first aimed to assess whether the delay-line-based fault injection was still feasible with the presence of an OS. The adopted method is equivalent to the one described in the bare-metal experiments 5.3.2.1, that is, a victim application loads an array from SDRAM while the attacker injects a delay glitch.

This time, memory caches L1 and L2 were enabled and the default OS kernel applications were running simultaneously to the experimental applications. To avoid array caching, the victim application directly instantiates the array within the SDRAM by mapping memory addresses using `mmap`. By doing so, the victim application is forced to load data from the SDRAM: `array_value = *(volatile int)(virt_SDRAM_addr)`.

The results below illustrate the array values loaded from SDRAM without and with the presence of a delay glitch (these array values were randomly initialized by the victim app). In this experiment, the attacker app used the delay-line controlling the strobe for the data byte #0. By XORing the results, we can indeed observe that the faults only impact the array byte #0. This observation was experimentally tested and confirmed on the 3 other data bytes and for both read and write attack directions.

```
[attacker] Byte 0, stress:  1, delay:  1620
[attacker] fault:  683dede1 -> 683dede3 - XOR: 00000002
[attacker] fault:  08bbb4a7 -> 08bbb4ef - XOR: 00000048
[attacker] fault:  61bef407 -> 61bef405 - XOR: 00000002
[attacker] fault:  34a787be -> 34a787b4 - XOR: 0000000a
```

The OS kernel is a potential collateral victim of our experiments. We observed that the kernel was sometimes subjected to memory errors such as page faults or NULL pointer dereference. These errors usually induce mutes and necessitate a board reset to restart the fault campaign. Despite this new constraint, it usually took less than 10 seconds to obtain a successful fault as long as the glitch and timing parameters were properly calibrated. To conclude, the fault model remains consistent with or without the presence of an OS although the latter is more likely to be subjected to mutes.

5.3.3.3 Persistent Fault Analysis Attack on AES

PFA fits particularly well with the adopted Linux setup as it relaxes the need for time-synchronization which cannot easily be controlled on Linux because of the timing jitter constantly induced by kernel interruptions. The evaluated AES executable implements the Tiny AES source code and takes the number of encryption requested as argument. We chose a number of 6,000 encryptions to conduct the PFA. For each launch, the application computed 6,000 AES encryptions using a predefined plaintext sequence and stored the associated ciphertexts in a text file. The presumably faulted output file was each time compared with a correct file in order to detect a divergence. If so, we saved it. Based on this setup, we conducted 10,000 injections ($\sim$ 2 hours) and obtained 26 files containing faulty ciphertexts.

All the 26 files contained results suitable for PFA. However, 25 of them were subjected to an additional difficulty brought by the OS which causes the faulty Sbox to get evicted from cache before the end of the encryption process. In these cases the faulty Sbox didn't remained cached during enough encryptions and the PFA attack couldn't be completed (PFA requires at least 1,641 faulty ciphertexts [146]). In this campaign, a unique file met the PFA requirements and the AES key was successfully retrieved using 4,375 faulty ciphertexts. Later on, we were able to reproduce this attack with different data and key.

5.3.3.4 Bellcore Attack on OpenSSL Signatures

Using the OpenSSL v1.1.0a version available on GitHub, we deployed a signature application written in C. This simple application reads an input text file, signs it using the OpenSSL `RSA_private_encrypt()` function [106] and stores the computed signature within an output file. The `rsa_ossl_mod_exp()` function used for modular exponentiation implements the Chinese Remainder Theorem (CRT) optimization [112]. CRT speeds up the RSA computation by splitting the signature calculation into two simpler partial signature computations which are combined afterward.

The Bellcore attack [15] demonstrates that a fault injection corrupting the calculation of a partial signature leaks the RSA private key. If the attacker is able to collect a correct signature S and a faulty signature S', he may retrieve the entire RSA private key by computing $gcd(S' - S, N)$. According to Listing 5.3, we conducted 1,500 injections on our RSA signature application and collected 45 faulty signatures. However, as in [100], none of them led to a successful Bellcore attack.

The `rsa_ossl_mod_exp()` function includes a protection against Bellcore attacks. Before outputting the computed signature, it verifies that $S^e - M = 0$. Where S is the computed signature, e the public exponent and M the file to sign. To check whether this verification was the cause of our failure, we removed it and recompiled OpenSSL. By injecting 100,000 glitches ($\sim$ 20 hours) on the unprotected version, we obtained 5,713 various errors including 14 faulty signatures . All of them led to RSA full private key

recovery. This result demonstrates that delay-line-based fault injection is not restricted to DFA and PFA but can also be used to conduct Bellcore attacks on unprotected RSA implementations.

On the protected OpenSSL version, we tried to inject double faults in order to generate a faulty signature and then bypass the protection. However, considering the fact that the success rate for a single fault was 0.00014%, a double fault event was very unlikely to occur. This limitation is discussed in Subsection 5.3.4.

5.3.3.5 Conclusion on OS Results

With an OS implemented, additional difficulties arise such as synchronization control or cache eviction constraints. Despite these restrictions, we demonstrated that data byte corruption, PFA and Bellcore fault attacks were feasible. The following section addresses the overall performance and limitations of our attack vector and compares it with existing works.

5.3.4 Discussion

5.3.4.1 Related Works

In 2014, Rowhammer [74, 57] was the first hardware vulnerability to be exploited using software programming only. Due to an undesirable side effect in DRAM memories, a malware could induce bit flips in DRAM without accessing it. This threat directly exposed millions of devices to hardware attacks. Inspired by the Rowhammer potential, the research works related to SbHA soared. In 2017, a new family of SbHA arose with ClkSCREW [128]. This work demonstrated that by manipulating power and frequency regulators using OS-kernel drivers, an attacker could inject power and frequency glitches inside processes. Again, this hardware attack vector bypassed software isolation and was used to break state-of-the-art security features such as TZ on ARM devices. This attack was also demonstrated against Software Guard Extensions (SGX) [67] on Intel processors with Voltjockey, Plundervolt and V0ltpwn [111, 100, 73]. In addition to SoCs, several temperature and voltage glitch attacks were conducted on FPGA and demonstrated the dangers of multi-tenants FPGA cloud datacenters [80]. The SbHA topic is growing fast and represents a frightening threat for connected device security. The increasing number of ethical disclosures, CVEs and security patches deployed by industrial underlines the extent of the threat.

5.3.4.2 Advantages and Limitations over Prior Methods

In the following paragraph, we list the strengths and weaknesses of delay-line-based fault injection by comparing it to other existing attack vectors.

Fault Privilege: Because it accesses hardware registers to inject the fault, *FaultLine*

may require root privileges depending on the implemented OS. Although being not as reproducible as the Rowhammer attack which can be launched from user-space, multiple real-world scenarios fit with its prerequisites. All modern systems where entities with different privileges share external memories are potential targets.

Fault Surface: 1) *FaultLine* affects all the memory accesses regardless of their origin. This phenomenon enables the fault injection on other processes and thus the attack. However, this could sometimes induce errors on untargeted processes. These collateral damages may cause mutes when applied to kernel processes.

2) Even if our experiments used the delay-lines located in the DDR memory controller, *FaultLine* should not be regarded as SDRAM specific and could be conducted on other components embedding delay-lines in the future (e.g Flash, SD and MMC memory controllers).

Fault Synchronization: As for typical glitch injection, *FaultLine* success depends on timing synchronization. This differs from Rowhammer whose success is more related to the bit flip location in DRAM. In *FaultLine*, the right timings were found through try-and-error using delay routines (`for` loops, hardware performance counters). This could be improved by the use of cache side-channel or SbSCA [53] to finely time the victim operations.

Fault Rate: The delay-line glitch success rate on memory transfers can be rounded to 0.1%. To improve it, one could inject wider glitches but would be submitted to more mutes. Even if V0LTpwn achieves an impressive 99% success rate on OpenSSL hash computations [73], this limited success ratio is recurrent in SoC attacks and exists in Rowhammer and Plundervolt (it takes around 500,000 iterations for Plundervolt to inject a multiplication error [100]).

Fault Precision: Delay-line registers can be finely tuned. They provide full control on the byte and the direction (read/write) chosen by the attacker. These parameters enable precise injection and facilitate attacks such as DFA. They are inherent to *FaultLine* and offer an improved fault control in comparison to voltage/frequency glitch injection methods.

5.3.4.3 Countermeasures

In the following, we describe three potential approaches to mitigate *FaultLine*.

Disabling access to delay-lines: As stated previously, synchronization components such as delay-lines cannot be removed and have to remain programmable. However, they should only be accessible during the DRAM training sequence at boot time. A simple mitigation could act at system level by preventing the access to the delay-line registers by unauthorized software entities. Hence, only the OS for instance would have access to this resource. Another method would be to use a TZ aware device that can prevent the delay-line access from the normal world users.

Fault-Resistant Software: As described in the OpenSSL RSA attack experiments, traditional software fault injection countermeasures such as signature verification may ef-

fectively mitigate *FaultLine*. Indeed, the limited *FaultLine* success rate prevents it from injecting double faults in a reasonable amount of time and thus from defeating this type of protections. However, this might not be true for all the existing countermeasures. For instance, detection based countermeasures such as duplicate encryption [124] may fail due to the persistent behavior of the fault injected (e.g PFA on AES). To remain effective, these mitigation methods would require to flush the cache between each encryption leading thus to significant time overheads.

Error-Correction Code (ECC): The use of software or hardware ECC stands as an efficient method to detect and correct some of the errors induced by *FaultLine*. By computing a function which depends on the data stored (e.g Hamming Code), the ECC mechanism is able to detect corrupted memory transfers. However, it was proven in [27] that ECC can be defeated with the injection of precise faults on both the data and the computed ECC function. Because our attack vector provide a fine control on the fault applied it could be employed as Rowhammer to defeat this kind of memory protection.

5.4 Conclusion on Delay-Line based Fault Injection

In this chapter, we evaluated the use of delay-lines-based components as a potential fault injection mechanism suitable for SbHA. By leveraging the open access to delay-lines in a Zynq processor, we were able to induce glitches in external memory accesses and more significantly to corrupt data transfers, RSA signatures and even to retrieve keys from cryptographic applications. Delay-line-based fault injection is unprecedented, it provides new controls and parameters that can be used to finely shape the applied glitch. It thus stands as a powerful novel alternative to the existing SbFIA mechanisms. Delay-lines are implemented in a wide range of electronic devices from microcontrollers to complex processors. Because malware exploitation of this threat could emerge in a near future, delay-lines should now be considered as a potential threat and systematically protected against malicious modifications.

5.4.1 Results Reminder

The experimental results described in this chapter were published in [51] with co-authors Jean-Max Dutertre, Yannick Teglia and Philippe Loubet Moundi under the name "Fault-Line: Software-based Fault Injection on Memory Transfers".

Responsible Disclosure: We responsibly disclosed our findings to Xilinx on March 4th, 2021 which acknowledged and agreed on the publication of these results. Please keep in mind that *FaultLine* was performed on the Zynq 7000 processor for demonstration purposes. The concept is generic and any devices that embed delay-lines for external memory access management can be affected.

5.4.1.1 Main Contributions

The works conducted in this chapter led to:

- The discovery of a novel SbFIA attack vector based on delay-lines widely implemented in modern SoC devices. In contrary to previous SbFIA attack vectors based on power glitch injection, *FaultLine* uses delay-glitch injection and cannot be detected by power glitch detectors.
- The implementation of traditional and novel fault injection techniques such as DFA and PFA on complex SoC systems running rich OS.

5.4.1.2 Some Numbers

Here we lay out some numbers to remind delay-line-based glitching performances.

Attack Results:

- A DFA attack on AES can succeed in less than **100,000 injections** in a baremetal setup. This attack takes **2.4 minutes** in practice.

- A PFA attack on AES can succeed in less than **1,000 injections** in a baremetal setup. This attack takes **10 seconds** in practice.
- A PFA attack on AES can succeed in less than **10,000 injections** in a rich OS setup. This attack takes **2 hours** in practice.
- A Bellcore attack on RSA can succeed in less than **7,100 injections** in a rich OS setup. This attack takes **1,4 hours** in practice.

5.4.2 *SbHA Knowledge*: Toward Large Scale SbHAs

Through the study of SbHAs, we aim at warning the community that even remote systems, that by nature were not supposed to be the target of hardware attacks, are now at risk. Every modern processor is potentially vulnerable. While our contribution focused on ARM devices, x86 exploits have already been studied and shown feasible in other publications [85].

Even if SbHAs make remote hardware attacks possible, they don't make them simpler. Mounting a SbHA usually requires more dedication than conducting a classical software attack. For this reason, it will probably be essentially targeting assets that cannot be easily attacked from software in the first place. When considering the massive adoption of smart-card-level security and HW-based TEE in recent SoC devices, this projection makes sense.

The installation of the SbHA malware can come from various remote and local vectors. On the remote side, an over-the-air installation is possible. In the technical documentation of the 2021 Pegasus spyware [55] it is mentioned that "A push message is remotely and covertly sent to the mobile device. This message triggers the device to download and install the malware agent on the device". The Pegasus spyware download that has been infecting thousands of phones does not require any consent from the victim and "cannot be prevented by the target". Other remote methods such as virus contained in e-mail attachments[1] or scripts hidden in malicious installation files can be adopted to propagate the malware. On the local side, the malware can be physically injected within the target devices before they are shipped to the consumers (man-in-the-middle attack). Another possibility is the malware injection through a removable media such as an infected USB flash drive [72] or a network card [34].

The experimentation conducted in *FaultLine* were launched from a remote computer connected through SSH to a victim board. With an SSH communication channel opened, we were able to load malicious scripts and test the attack feasibility. In practice, large scale SbHA will be facilitated by the wide variety of existing methods that can lead to the execution of a malicious code on a remote device. To increase attack scalability, the malware scripts will need to scan the architecture under attack (OS in use, attack vector available, register addresses, etc) and adjust their parameters to enable the exploit.

Through our works on SbHA, we participated in demonstrating that modern devices

[1] https://en.wikipedia.org/wiki/ILOVEYOU

are highly susceptible to SbSCA and SbFIA attacks. This study led to the recognition of the SbHA threat on both academic and industrial worlds. To prevent their proliferation in the future, we believe that IC manufacturers and OS developers must rapidly allocate additional resources on detecting flaws within the software-hardware cooperation. This topic is discussed in the chapter 6.

Chapter 6. Conclusion and Perspectives

This thesis aimed at assessing emerging remote hardware attack techniques and at forecasting the extent of the Software-based Hardware Attack (SbHA) threat. Throughout this manuscript, we classified SbHA families and pointed out their similarities and differences with traditional local hardware attacks. Moreover, we contributed in discovering novel attack vectors that could be remotely controlled to conduct Software-based Side-Channel Analysis (SbSCA) and Software-based Fault Injection Attack (SbFIA) on connected devices. By building complex SbHAs, we aimed at promoting their recognition. This acknowledgment is essential to trigger the security community interest and prepare the future of hardware defenses. Indeed, if nothing is done in a near future to prevent attackers from accessing devices hardware resources, SbHAs could become a major threat for cyber-security. Even more detrimental, the multiplication of connected devices and their increasing complexity will probably facilitate their proliferation. In this chapter, we summarize the findings that emerged from our research on SbHAs and discuss their impact for the present and the future of hardware security.

Chapter Contents

6 Conclusion and Perspectives **153**

6.1 Manuscript Summary 154

6.2 Conclusion on SbHA 155

6.3 Thesis Impact 156

6.4 SbHA Perspectives 157

6.1 Manuscript Summary

This thesis evaluated SbHA as a new security threat for connected systems. In this section, we summarize the contributions of this manuscript.

In Chapter 2, we studied and defined new remote hardware threats. We classified these attacks in three distinct groups: TEMPEST attacks, Microarchitectural attacks and SbHA and highlighted major differences between those. As this thesis focused on SbHA, we provided an up-to-date classification containing many conference proceedings, journal articles and blog posts that can be attributed to this attack family. Moreover, we added SbHA to the traditional hardware attack classification (non-invasive, semi-invasive, invasive attacks) since it uses the same attack mechanisms but does not fit in any existing group. Finally, we demonstrated how a SbHA differs from a local hardware attack as it does not require a physical access to the target device. This major difference completely changes the attack paradigm since an SbHA can be launched remotely and simultaneously on thousands of connected devices without the victims knowing they are under attack. Therefore, a SbHA could last for days, months or years until the adversary obtains enough information to retrieve the victim's secrets. Moreover, the systems that were thought to be immune to hardware attacks (because physically inaccessible) are now potentially at risk with the advent of SbHAs.

In Chapter 3, we conducted our first SbHA experimentations on Field-Programmable Gate Array (FPGA). By leveraging the reconfigurable logic, we were able to deploy delay-based sensors suitable for collecting on-chip voltage. The emergence of FPGA in heterogeneous devices and their adoption in multi-user cloud systems introduced various threat models where a sensor located in the fabric aims at eavesdropping the activity of other users or assets. We deployed these attack scenarios from an FPGA block to another (FPGA-to-FPGA attack) and also from an FPGA block to a Computer Processing Unit (CPU) located on the same silicon die (FPGA-to-CPU attack). These research works eventually led to the creation of a high-speed FPGA-based delay-sensor dedicated to power Side-Channel Analysis (SCA) applications, to the first statistical SCA attacks conducted from an FPGA to a CPU and to the conception of an open-source framework aiming at facilitating FPGA-based SCA analysis.

In Chapter 4, we leveraged the knowledge drawn from the previous FPGA-based works to mount SbSCA attacks on System-on-Chip (SoC) devices. We discovered that delay-line components widely implemented in modern device memory controllers were suitable for voltage sensing. Based on these elements, we conducted the first Application Processor (AP)-vs-AP and AP-vs-MCU SbSCA attacks ever done on complex devices and called the exploit *SideLine*. Additionally to these SCA results obtained with these unconventional sensors, this work also described a detailed method for identifying SbHA vectors and conducting the attacks from a software program running within the target.

Finally, Chapter 5 introduced *FaultLine* a new SbFIA vector that affects a wide range of SoC devices using external memories. This attack medium also built on delay-line

has the particularity to target external memory transfers. Based on a malicious program, the delay-line can be programmed to inject glitches on external memory read and write operations. Using this unconventional attack vector we were able to mount advanced fault injection attacks such as DFA and PFA and break the isolation between adversary and victim applications running on rich Operating System (OS) environments.

6.2 Conclusion on SbHA

In the introduction of this manuscript we identified three major thesis objectives: the study of SbHA feasibility, the identification of new SbHA vectors and the exploration of new SbHA scenarios. These objectives aimed at answering our main research question: should we consider SbHA as a serious threat for Integrated Circuit (IC) security?

The research works described throughout this manuscript attempted to fulfill these three objectives. We evaluated the feasibility of SbHA on various devices (CPU, MCU, FPGA), we discovered new SbFIA and SbSCA vectors through the identification of delay-lines, and finally, we used these vectors to conduct new SbHA scenarios as described in the previous section. Through these experiments we partly answered the research question by demonstrating that SbFIA and SbSCA attacks were indeed capable of extracting secrets from a device but also relevant in a wide variety of connected systems.

Additionally to the obtained experimental results, we complemented our research work with various descriptions of the SbHA properties that were discovered throughout the experiments. These observations and attack techniques were described at the end of each experimentation chapter. In the reminder list below, we recall the main SbHA properties discussed in this manuscript.

- **SbHA Concept** *(Chapter 2)*: A SbHA attempts to inject and run a malicious program within a target device to access software-exposed hardware mechanisms (sensors, actuators) and conduct physical attacks such as Fault Injection Attack (FIA) or SCA.

- **SbHA versus Local Attacks** *(Chapter 2)*: A SbHA differs from an usual hardware attack since it does not involve any equipment beyond the target itself. In that way, it enables remote hardware attacks on connected devices.

- **SbHA Origins** *(Chapter 2)*: SbHAs are facilitated by the increasing complexity and connectivity of modern ICs. Various SoC hardware components are left accessible from software and are now efficient and flexible enough to be used for SbSCA and SbFIA attacks. Moreover, the massive adoption of multi-user systems, multi-security domains and integrated security features acts as a catalyst for the SbHA proliferation.

- **SbHA Vectors** *(Chapter 2, 3, 4, 5)*: SbHAs leverage hardware components implemented in complex ICs that can alter the behavior of an asset (SbFIA) or eavesdrop

its activity (SbSCA). Various SoC components can be maliciously used to inject errors (regulators, delay-lines, FPGA-based viruses) as well as collecting side-channels (voltage sensors, ADCs, delay-lines, FPGA-based sensors). SbHA vector identification is conducted through the analysis of user exposed OS commands, the analysis of target's hardware documentation or the reversing of OS device drivers.

- **SbHA Targets and Privileges** *(Chapter 2, 5)*: Depending on the adversary malware privilege level, different targets may be evaluated. An unprivileged adversary may try to bypass the logical isolation between processes or perform OS privilege escalation. However, he may be limited to the SbHA vectors that are accessible with unprivileged access rights (e.g. Rowhammer). A privileged adversary may access any SbHA vector (ADC, delay-line, regulators) but will be limited to the evaluation of higher privilege levels targets. That is, integrated security features such as hypervisors, hardware-based Trusted Execution Environments (TEEs) or secure elements.

- **SbHA Feasibility** *(Chapter 3, 4, 5)*: All the attack methodologies and techniques that enable SbHA exploits such as vector access, SCA samples alignment, data storage, data export, traces and processes synchronization are described throughout the chapters of this thesis. By analyzing and identifying the adversary tools enabling SbHAs, we aimed at facilitating the implementation of appropriate software and hardware countermeasures.

Finally, according to our experimental results and to the other SbHA works published recently, it appears that any door left open allowing a malware to gain access to hardware resources (unprivileged or privileged) may lead to a complete disclosure of a target's content. Therefore, there is an urgent need to build on the lessons learned from this thesis and employ countermeasures to effectively mitigate SbHAs.

6.3 Thesis Impact

This thesis conducted under a partnership between academic and industrial actors produced significant results on both sides.

On the academic side, the novelty of the experimentation conducted led to the publication of several scientific communications. A total of four conference papers; two on FPGA-based SCA [50, 54], one on delay-line-based SbSCA (*SideLine*) [53] and one on delay-line-based SbFIA (*FaultLine*) [51] were published in hardware security and FPGA conferences around the world (CARDIS 2019, ReConFig 2019, COSADE 2021, HOST 2021). These works were also presented in various workshops throughout France and Europe such as the 2019 Asset meeting in Munich, the 2020 iMath seminar in Toulon and the 2021 DGA INRIA seminar in Rennes.

On the industrial side, our discoveries led to the application of two patents (still in review at this point). Moreover, our works on SbHA were repeatedly presented in internal

workshops and led to the collaboration with various Thales business lines to evaluate the remote hardware attack threat on several products. Finally, our research works were presented at the JIL Hardware-related Attacks Subgroup (JHAS), a conglomerate of security evaluation laboratories, smart-card/IC manufacturers and national certification authorities. JHAS actors recognized the SbHA threat and this communication contributed to the adoption of SbHA in Common Criteria evaluation methodology for smart cards and similar devices. By consequence it will be automatically included in the new catalog of attacks that will be considered for evaluations under the future European Union Certification scheme.

6.4 SbHA Perspectives

In the recent years, the influx of connected devices has been modifying every aspects of our lives. Even if this revolution improves our standards of livings, the interconnection between objects should not question the data privacy. While cyber-attacks have often been taking advantage of software vulnerabilities, we've seen in this thesis that software-accessible hardware flaws stand as major threats for today's connected systems.

Various perspectives could arise from this thesis work. The FPGA evaluation conducted in 3 demonstrated the eventuality of SbSCA attacks on multi-tenant FPGA platforms. Additionally to cryptographic secrets retrieval, these attacks could be applied on other targets such as machine learning accelerators and used to clone neural models. The *SideLine* and *FaultLine* vulnerabilities presented in chapter 4 and 5 could be evaluated on other platforms such as Intel or RISC V devices. This would pave the way to new attack scenarios targeting security features such as RISC V MultiZone[1] and Intel SGX. Moreover, while our experimentations focused on delay-lines blocks and delay-locked-loops, other integrated components such as phase-locked-loops, voltage sensors and temperature sensors could be evaluated. Finally, we believe that the current SbHA vectors discovered are only the tip of the iceberg. With the integration of new hardware resources such as neural units, analog logic and reconfigurable logic in complex SoCs, various sensing and faulting mechanisms could be discovered and maliciously employed in a near future.

Remote hardware attack is a wide topic that covers timing, micro-architectural, TEMPEST and software-based families. In this thesis, we narrowed our research to the SbHA field that is the most similar to local hardware attacks conducted in laboratories. At this point, this recent attack family initiated by Rowhammer in 2014 has certainly not demonstrated all its potential. While our researches described its ability to bypass logical isolation in complex devices and using novel attack mediums; multiple unexplored scenarios remain open. We believe that integrated security components such as integrated-SIM, cryptographic accelerators and secure elements in general will be the next SbHA targets. For this reason, we urge secure IC manufacturers to strengthen their products against SbHA since successful exploits could affect millions of devices and provoke major secu-

[1] https://hex-five.com/multizone-security-sdk/

rity crises.

Thankfully, the fast JHAS SbHA recognition and the starting of threat assessment by security evaluation laboratories could prevent the worst from happening. Moreover, the works conducted during this thesis demonstrated that SbHA could be mitigated in various ways. For instance, we observed that traditional SCA and FIA countermeasures already implemented in smart-cards and secure ICs remained efficient against SbHAs. Their implementation should be generalized to any connected device potentially affected by SbHAs (e.g multi-user systems, devices with different security privileges). Moreover, each contribution done on SbHAs came with countermeasure guidelines that need to be implemented by IC manufacturers and OS developers to prevent attackers from accessing the hardware vectors.

The SbHA study is a growing topic that will persist over time. As it requires both skills in programming and hardware hacking, it will create a stronger junction between the hardware and software security communities. And this is for the best. Indeed, the historic lack of communication between these two entities participated in the implementation of security oblivious software-accessible hardware mechanisms which are partly responsible for the emergence of SbHAs. From now on, software and hardware designers will need to work hand in hand to effectively and persistently mitigate the SbHA threat. Successful or not, future will tell.

Bibliography

[1] Bilal I Abdulrazzaq, Izhal Abdul Halin, Shoji Kawahito, et al. "A review on high-resolution CMOS delay lines: towards sub-picosecond jitter performance". In: *SpringerPlus* 5.1 (Dec. 2016), p. 434. DOI: 10.1186/s40064-016-2090-z.

[2] Michel Agoyan, Jean-Max Dutertre, David Naccache, et al. "When Clocks Fail: On Critical Paths and Clock Faults". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6035 LNCS. 2010, pp. 182–193. DOI: 10.1007/978-3-642-12510-2_13.

[3] Md Mahbub Alam, Shahin Tajik, Fatemeh Ganji, et al. "RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions". In: *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, Aug. 2019, pp. 48–55. DOI: 10.1109/FDTC.2019.00015.

[4] Altera Corporation. "What is an SoC FPGA?" In: *Altera Support Resources* (2014), pp. 1–4. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ab/ab1%7B%5C_%7Dsoc%7B%5C_%7Dfpga.pdf.

[5] AMD. *AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More*. Tech. rep. 2020, pp. 1–20. URL: https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[6] Ross Anderson and Markus Kuhn. "Low cost attacks on tamper resistant devices". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1361. 1998, pp. 125–136. DOI: 10.1007/BFb0028165.

[7] Limited ARM. *ARM PrimeCell MultiPort Memory Controller (PL176) Technical Reference Manual*. Tech. rep. 2003. URL: https://developer.arm.com/documentation/ddi0269/a/dynamic-memory-controller.

[8] Limited ARM. *Building a Secure System using TrustZone Technology*. Tech. rep. 2008. URL: https://developer.arm.com/documentation/genc009492/c.

[9] Limited ARM. *TrustZone technology for Armv8-M Architecture*. Tech. rep. 2016, pp. 1–28. URL: https://developer.arm.com/documentation/100690/0201/Preface/About-this-book.

[10] Michael Backes, Markus Dürmuth, Sebastian Gerling, et al. “Acoustic side-channel attacks on printers”. In: *Proceedings of the 19th USENIX Security Symposium*. 2010, pp. 307–322. DOI: 10.5555/1929820.1929847.

[11] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. “Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 7779 LNCS. 2013, pp. 1–17. DOI: 10.1007/978-3-642-36095-4_1.

[12] Erick Bauman and Zhiqiang Lin. “A Case for Protecting Computer Games With SGX”. In: *Proceedings of the 1st Workshop on System Software for Trusted Execution*. New York, NY, USA: ACM, Dec. 2016, pp. 1–6. DOI: 10.1145/3007788.3007792.

[13] Eli Biham and Adi Shamir. “Differential fault analysis of secret key cryptosystems”. In: *Advances in Cryptology — CRYPTO ’97*. Ed. by Burton S Kaliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 513–525. DOI: 10.1007/BFb0052259.

[14] A Bogdanov, L R Knudsen, G Leander, et al. “PRESENT: An Ultra-Lightweight Block Cipher”. In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31.

[15] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. “On the Importance of Checking Cryptographic Protocols for Faults”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1233. 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4.

[16] Andrew Boutros, Mathew Hall, Nicolas Papernot, and Vaughn Betz. “Neighbors From Hell: Voltage Attacks Against Deep Learning Accelerators on Multi-Tenant FPGAs”. In: *2020 International Conference on Field-Programmable Technology (ICFPT)* abs/2012.0 (Dec. 2020), pp. 103–111. DOI: 10.1109/ICFPT51103.2020.00023.

[17] Ferdinand Brasser, Lucas Davi, David Gens, et al. “Can’t Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory”. In: *Proceedings of the 26th USENIX Conference on Security Symposium*. Vancouver, BC: USENIX Association, 2017, pp. 117–130. DOI: 10.5555/3241189.3241200.

[18] Eric Brier, Christophe Clavier, and Francis Olivier. “Correlation Power Analysis with a Leakage Model”. In: *Cryptographic Hardware and Embedded Systems*. 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2.

[19] David Brumley and Dan Boneh. “Remote timing attacks are practical”. In: *Computer Networks* 48.5 (Aug. 2005), pp. 701–716. DOI: 10.1016/j.comnet.2005.01.010.

[20] Sebanjila Kevin Bukasa, Ronan Lashermes, Hélène Le Bouder, et al. “How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip”. In: *Lecture Notes in Computer Science*. 2018, pp. 93–109. DOI: 10.1007/978-3-319-93524-9_6.

[21] Giovanni Camurati, Aurélien Francillon, and François-Xavier Standaert. “Understanding Screaming Channels: From a Detailed Analysis to Improved Attacks”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (June 2020), pp. 358–401. DOI: 10.46586/tches.v2020.i3.358-401.

[22] Giovanni Camurati, Sebastian Poeplau, Marius Muench, et al. “Screaming Channels”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’18. New York, NY, USA: ACM, Oct. 2018, pp. 163–177. DOI: 10.1145/3243734.3243802.

[23] Claudio Canella, Daniel Genkin, Lukas Giner, et al. “Fallout”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, Nov. 2019, pp. 769–784. DOI: 10.1145/3319535.3363219.

[24] Fei Chen, Yi Shan, Yu Zhang, et al. “Enabling FPGAs in the cloud”. In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. New York, NY, USA: ACM, May 2014, pp. 1–10. DOI: 10.1145/2597917.2597929.

[25] Ching-che Chung, Pao-lung Chen, and Chen-yi Lee. “An All-Digital Delay-Locked Loop for DDR SDRAM Controller Applications”. In: *2006 International Symposium on VLSI Design, Automation and Test*. IEEE, Apr. 2006, pp. 1–4. DOI: 10.1109/VDAT.2006.258159.

[26] Christophe Clavier, Benoit Feix, Georges Gagnerot, et al. “Horizontal Correlation Analysis on Exponentiation”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6476 LNCS. 2010, pp. 46–61. DOI: 10.1007/978-3-642-17650-0_5.

[27] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. “Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019, pp. 55–71. DOI: 10.1109/SP.2019.00089.

[28] Jean-Sébastien Coron. “Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems”. In: *Cryptographic Hardware and Embedded Systems*. Ed. by Çetin K Koç and Christof Paar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 292–302.

[29] Nicolas T. Courtois. *Hacking Pay-TV*. 2008. URL: http://www.nicolascourtois.com/papers/sc/PayTv.pdf.

[30] Joan Daemen and Vincent Rijmen. “The Block Cipher Rijndael”. In: *Smart Card Research and Applications*. 2000, pp. 277–284. DOI: 10.1007/10721064_26.

[31] Shidhartha Das, Paul Whatmough, and David Bull. “Modeling and characterization of the system-level Power Delivery Network for a dual-core ARM Cortex-A57 cluster in 28nm CMOS”. In: *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, July 2015, pp. 146–151. DOI: 10.1109/ISLPED.2015.7273505.

[32] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. “Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES”. In: *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, Sept. 2012, pp. 7–15. DOI: 10.1109/FDTC.2012.15.

[33] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, et al. “Injection of transient faults using electromagnetic pulses Practical results on a cryptographic system”. In: *Cryptology ePrint Archive* (2012). URL: https://hal.archives-ouvertes.fr/emse-00742850/en.

[34] Loïc Duflot, Yves-Alexis Perez, and Benjamin Morin. “What If You Can’t Trust Your Network Card?” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6961 LNCS. 2011, pp. 378–397. DOI: 10.1007/978-3-642-23644-0_20.

[35] Jean-Max Dutertre, Bruno Robisson, Assia Tria, and Loic Zussa. “Investigation of timing constraints violation as a fault injection means”. In: *Design of Circuits and Integrated Systems* (2012). URL: https://hal.archives-ouvertes.fr/emse-00742652.

[36] Wim van Eck. “Electromagnetic radiation from video display units: An eavesdropping risk?” In: *Computers & Security* 4.4 (Dec. 1985), pp. 269–286. DOI: 10.1016/0167-4048(85)90046-X.

[37] Maik Ender, Amir Moradi, and Christof Paar. “The unpatchable silicon: A full break of the bitstream encryption of Xilinx 7-series FPGAS”. In: *Proceedings of the 29th USENIX Security Symposium* (2020), pp. 1803–1819. URL: https://www.usenix.org/system/files/sec20-ender.pdf.

[38] Daisuke Fujimoto, Noriyuki Miura, Makoto Nagata, et al. “On-Chip Power Noise Measurements of Cryptographic VLSI Circuits and Interpretation for Side-Channel Analysis”. In: *International Symposium on Electromagnetic Compatibility (EMC Europe)*. Brugge, Belgium: IEEE, 2013, pp. 405–410. URL: https://ieeexplore.ieee.org/document/6653337.

[39] Toshinori Fukunaga and Junko Takahashi. “Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers”. In: *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, Sept. 2009, pp. 84–92. DOI: 10.1109/FDTC.2009.34.

[40] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. “Electromagnetic Analysis: Concrete Results”. In: *Cryptographic Hardware and Embedded Systems*. 2001, pp. 251–261. DOI: 10.1007/3-540-44709-1_21.

[41] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. “A survey of microarchitectural timing attacks and countermeasures on contemporary hardware”. In: *Journal of Cryptographic Engineering* 8.1 (Apr. 2018), pp. 1–27. DOI: 10.1007/s13389-016-0141-6.

[42] Daniel Genkin, Adi Shamir, and Eran Tromer. “RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis”. In: *International Journal of Knowledge-Based Intelligent Engineering Systems vol.* Vol. 3, no.3, J. 2014, pp. 444–461. DOI: 10.1007/978-3-662-44371-2_25.

[43] Ilias Giechaskiel, Kasper B. Rasmussen, and Ken Eguro. “Leaky Wires”. In: *Asia Conference on Computer and Communications Security*. 2018. DOI: 10.1145/3196494.3196518.

[44] Ilias Giechaskiel, Kasper Bonne Rasmussen, and Jakub Szefer. “C3APSULe: Cross-FPGA Covert-Channel Attacks through Power Supply Unit Leakage”. In: *2020 IEEE Symposium on Security and Privacy (SP)* (May 2020), pp. 1728–1741. DOI: 10.1109/SP40000.2020.00070.

[45] Ognjen Glamocanin, Louis Coulon, Francesco Regazzoni, and Mirjana Stojilovic. “Are Cloud FPGAs Really Vulnerable to Power Analysis Attacks?” In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2020, pp. 1007–1010. DOI: 10.23919/DATE48585.2020.9116481.

[46] Dennis R. E. Gnad, Jonas Krautter, and Mehdi B. Tahoori. “Leaky Noise : New Side-Channel Attack Vectors in Mixed-Signal IoT Devices”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019). DOI: 10.13154/tches.v2019.i3.305-339.

[47] Dennis R. E. Gnad, Fabian Oboril, Saman Kiamehr, and Mehdi B. Tahoori. “An Experimental Evaluation and Analysis of Transient Voltage Fluctuations in FPGAs”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.10 (Oct. 2018), pp. 1817–1830. DOI: 10.1109/TVLSI.2018.2848460.

[48] Dennis R. E. Gnad, Fabian Oboril, and Mehdi B. Tahoori. “Voltage drop-based fault attacks on FPGAs using valid bitstreams”. In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Sept. 2017, pp. 1–7. DOI: 10.23919/FPL.2017.8056840.

[49] Dennis R. E. Gnad, Sascha Rapp, Jonas Krautter, and Mehdi B. Tahoori. “Checking for Electrical Level Security Threats in Bitstreams for Multi-tenant FPGAs”. In: *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, Dec. 2018, pp. 286–289. DOI: 10.1109/FPT.2018.00055.

[50] Joseph Gravellier, Jean-max Dutertre, Yannick Teglia, and Philippe Loubet-Moundi. “High-Speed Ring Oscillator based Sensors for Remote Side-Channel Attacks on FPGAs”. In: *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, Dec. 2019, pp. 1–8. DOI: 10.1109/ReConFig48160.2019.8994789.

[51] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, and Philippe Loubet Moundi. “FaultLine : Software-based Fault Injection on Memory Transfers”. In: *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2021.

[52] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, and Philippe Loubet Moundi. “SCAbox : A Framework for Facilitating FPGA-based Side-Channel Analysis”. In: (2021). URL: https://emse-sas-lab.github.io/SCAbox/.

[53] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, and Philippe Loubet Moundi. “SideLine: How Delay-Lines (May) Leak Secrets from your SoC”. In: *Constructive Side-Channel Analysis and Secure Design*. 2021. URL: http://arxiv.org/abs/2009.07773.

[54] Joseph Gravellier, Jean-max Dutertre, Yannick Teglia, et al. “Remote Side-Channel Attacks on Heterogeneous SoC”. In: *18th Smart Card Research and Advanced Application Conference*. 2020, pp. 109–125. DOI: 10.1007/978-3-030-42068-0_7.

[55] NSO Group. *Pegasus – Product Description*. Tech. rep. URL: https://s3.documentcloud.org/documents/4599753/NSO-Pegasus.pdf.

[56] Daniel Gruss, Moritz Lipp, Michael Schwarz, et al. “Another Flip in the Wall of Rowhammer Defenses”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2018, pp. 245–261. DOI: 10.1109/SP.2018.00031.

[57] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA 2016, 13th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. 2016, pp. 300–321. DOI: 10.1007/978-3-319-40667-1_15.

[58] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. "Flush+Flush: A Fast and Stealthy Cache Attack". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9721. 2016, pp. 279–299. DOI: 10.1007/978-3-319-40667-1_14.

[59] Meeta S. Gupta, Jarod L. Oatley, Russ Joseph, et al. "Understanding Voltage Variations in Chip Multiprocessors using a Distributed Power-Delivery Network". In: *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, Apr. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364663.

[60] Mordechai Guri, Assaf Kachlon, Ofer Hasson, et al. "GSMem : Data Exfiltration from Air-Gapped Computers over GSM Frequencies". In: *Proceedings of the 24th USENIX Security Symposium* (2015), pp. 849–864. DOI: 10.5555/2831143.2831197.

[61] Mordechai Guri, Yosef Solewicz, Andrey Daidakulov, and Yuval Elovici. "Fansmitter: Acoustic Data Exfiltration from (Speakerless) Air-Gapped Computers". In: *arXiv* (2016). URL: https://arxiv.org/abs/1606.05915.

[62] Mordechai Guri, Boris Zadov, and Yuval Elovici. "LED-it-GO: Leaking (A Lot of) Data from Air-Gapped Computers via the (Small) Hard Drive LED". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10327 LNCS. 2017, pp. 161–184. DOI: 10.1007/978-3-319-60876-1_8.

[63] Helena Handschuh, Pascal Paillier, and Jacques Stern. "Probing Attacks On Tamper-Resistant Devices". In: *Cryptographic Hardware and Embedded Systems - CHES 1999*. Vol. 1717. Lecture Notes in Computer Science. Springer, 1999, pp. 303–315. DOI: 10.1007/3-540-48059-5_26.

[64] Michael Hutter and Jörn-Marc Schmidt. "The Temperature Side Channel and Heating Fault Attacks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8419 LNCS. 2014, pp. 219–235. DOI: 10.1007/978-3-319-08302-5_15.

[65] Diligent Inc. *Zybo FPGA Board Reference Manual*. 2016. URL: https://reference.digilentinc.com/%7B%5C_%7Dmedia/zybo:zybo%7B%5C_%7Drm.pdf.

[66] Mehmet Sinan İnci, Berk Gulmezoglu, Gorka Irazoqui, et al. "Cache Attacks Enable Bulk Key Recovery on the Cloud". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9813 LNCS. 2016, pp. 368–388. DOI: 10.1007/978-3-662-53140-2_18.

[67] Intel. *Software Guard Extensions Programming Reference*. Tech. rep. 2014, p. 156. URL: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[68] MS-ISAC. *Ransomware Guide*. Tech. rep. September. 2020, pp. 1–16. URL: https://www.cisa.gov/sites/default/files/publications/CISA%7B%5C_%7DMS-ISAC%7B%5C_%7DRansomware%20Guide%7B%5C_%7DS508C.pdf.

[69] Mohammad A. Islam, Shaolei Ren, and Adam Wierman. "Exploiting a Thermal Side Channel for Power Attacks in Multi-Tenant Data Centers". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, Oct. 2017, pp. 1079–1094. DOI: 10.1145/3133956.3133994.

[70] Marc Joye and Sung Ming Yen. "The Montgomery Powering Ladder". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2523 (2003), pp. 291–302. DOI: 10.1007/3-540-36400-5_22.

[71] Julian E. Barnes. *Pentagon computer networks attacked*. 2008. URL: https://www.latimes.com/archives/la-xpm-2008-nov-28-na-cyberattack28-story.html.

[72] Stamatis Karnouskos. "Stuxnet worm impact on industrial cyber-physical system security". In: *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Nov. 2011, pp. 4490–4494. DOI: 10.1109/IECON.2011.6120048.

[73] Zijo Kenjar, Tommaso Frassetto, David Gens, et al. "V0LTpwn: Attacking x86 Processor Integrity from Software". In: *29th USENIX Security Symposium (USENIX Security 20)* (2020), pp. 1445–1461. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/kenjar.

[74] Yoongu Kim, Ross Daly, Jeremie Kim, et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, June 2014, pp. 361–372. DOI: 10.1109/ISCA.2014.6853210.

[75] Paul Kocher, Jann Horn, Anders Fogh, et al. "Spectre Attacks: Exploiting Speculative Execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. May 2019. DOI: 10.1109/SP.2019.00002.

[76] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 1666. 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[77] Paul C Kocher. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems”. In: *CRYPTO ’96*. 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[78] Kokke. *Tiny-AES-c*. 2018. URL: https://github.com/kokke/tiny-AES-c.

[79] Oliver Kömmerling and Markus G Kuhn. “Design Principles for Tamper-Resistant Smartcard Processors”. In: *Proceedings of the USENIX workshop on smartcard technology*. 1999. DOI: 10.5555/1267115.1267117.

[80] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. “FPGAhammer : Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 3 (2018), pp. 44–68. DOI: 10.13154/tches.v2018.i3.44-68.

[81] Jonas Krautter, Dennis R.E. Gnad, Falk Schellenberg, et al. “Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs”. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, Nov. 2019, pp. 1–8. DOI: 10.1109/ICCAD45719.2019.8942094.

[82] Markus G Kuhn. “Attacks on Pay-TV access control systems”. In: *Security Seminar, Computer Laboratory, Cambridge* (1997). URL: http://128.232.0.20/%7B~%7Dmgk25/vc-slides.pdf.

[83] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. “RAMBleed: Reading Bits in Memory Without Accessing Them”. In: *2020 IEEE Symposium on Security and Privacy (SP)*. May. IEEE, May 2020, pp. 695–711. DOI: 10.1109/SP40000.2020.00020.

[84] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, et al. “Armageddon: Cache attacks on mobile devices”. In: *Proceedings of the 25th USENIX Security Symposium* (2016), pp. 549–564. DOI: 10.5281/zenodo.59889.

[85] Moritz Lipp, Andreas Kogler, David Oswald, et al. “PLATYPUS: Software-based Power Side-Channel Attacks on x86”. In: *IEEE Symposium on Security and Privacy (SP)*. 2021. DOI: 10.1109/SP40001.2021.00063.

[86] Moritz Lipp, Michael Schwarz, Daniel Gruss, et al. “Meltdown: Reading Kernel Memory from User Space”. In: *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association, Jan. 2018, pp. 973–990. DOI: 10.5555/3277203.3277276.

[87] Moritz Lipp, Michael Schwarz, Lukas Raab, et al. “Nethammer: Inducing Rowhammer Faults through Network Requests”. In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*. 2020, pp. 710–719. DOI: 10.1109/EuroSPW51379.2020.00102.

[88] Heiko Lohrke, Shahin Tajik, Thilo Krachenfels, et al. “Key Extraction Using Thermal Laser Stimulation A Case Study on Xilinx Ultrascale FPGAs”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, Issu.3 (2018), pp. 573–595. DOI: 10.13154/tches.v2018.i3.573-595.

[89] Adrien Le Masle and Wayne Luk. “Detecting power attacks on reconfigurable hardware”. In: *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Aug. 2012, pp. 14–19. DOI: 10.1109/FPL.2012.6339235.

[90] Philippe Maurine, Karim Tobich, Thomas Ordas, and Pierre-Yvan Liardet. “Yet Another Fault Injection Technique : by Forward Body Biasing Injection”. In: *YACC’2012: Yet Another Conference on Cryptography* (2012). URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00762035.

[91] Thomas S Messerges. “Using Second-Order Power Analysis to Attack DPA Resistant Software”. In: *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. CHES ’00. Berlin, Heidelberg: Springer-Verlag, 2000, pp. 238–251. DOI: 10.1007/3-540-44499-8_19.

[92] Hassen Mestiri, Noura Benhadjyoussef, Mohsen Machhout, and Rached Tourki. “A Comparative Study of Power Consumption Models for CPA Attack”. In: *International Journal of Computer Network and Information Security* 5.3 (Mar. 2012), pp. 25–31. DOI: 10.5815/ijcnis.2013.03.03.

[93] ST Microelectronics. *STM32MP1 Reference manual*. Tech. rep. 2019. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32mp1-series.html.

[94] Shayan Moini, Shanquan Tian, Daniel Holcomb, et al. “Power Side-Channel Attacks on BNN Accelerators in Remote FPGAs”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.2 (June 2021), pp. 357–370. DOI: 10.1109/JETCAS.2021.3074608.

[95] Peter L. Montgomery. “Speeding the Pollard and Elliptic Curve Methods of Factorization”. In: *Mathematics of Computation* 48.177 (1987), p. 243. DOI: 10.2307/2007888.

[96] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. “On the vulnerability of FPGA bitstream encryption against power analysis attacks”. In: *Proceedings of the 18th ACM conference on Computer and communications security - CCS ’11*. New York, New York, USA: ACM Press, 2011, p. 111. DOI: 10.1145/2046707.2046722.

[97] Amir Moradi, David Oswald, Christof Paar, and Pawel Swierczynski. “Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II”. In: *Proceedings of the ACM/SIGDA international symposium on Field pro-*

grammable gate arrays - FPGA '13. New York, New York, USA: ACM Press, 2013, p. 91. DOI: 10.1145/2435264.2435282.

[98] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, et al. "Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller". In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, Aug. 2013, pp. 77–88. DOI: 10.1109/FDTC.2013.9.

[99] Motorola Inc. "M68000 8-/16-32-Bit Microprocessors User's Manual". In: (1991), p. 224. URL: https://www.nxp.com/docs/en/reference-manual/MC68000UM.pdf.

[100] Kit Murdock, David Oswald, Flavio D Garcia, et al. "Plundervolt: Software-based Fault Injection Attacks against Intel SGX". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2020, pp. 1466–1482. DOI: 10.1109/SP40000.2020.00057.

[101] National Security Agency (NSA). *Ghidra*. 2019. URL: https://ghidra-sre.org/.

[102] National Security Agency (NSA). "Tempest: A signal problem". In: 2.3 (1972). URL: https://cryptome.org/nsa-tempest.pdf.

[103] Colin O'Flynn and Zhizhang Chen. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Vol. 8622 LNCS. 2014, pp. 243–260. DOI: 10.1007/978-3-319-10175-0_17.

[104] Colin O'Flynn and Alex Dewar. "On-Device Power Analysis Across Hardware Security Domains.: Stop Hitting Yourself." In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019). DOI: 10.13154/tches.v2019.i4.126-153.

[105] T.J. O'Gorman. "The effect of cosmic rays on the soft error rate of a DRAM at ground level". In: *IEEE Transactions on Electron Devices* 41.4 (Apr. 1994), pp. 553–557. DOI: 10.1109/16.278509.

[106] OpenSSL Foundation. *OpenSSL*. 2002. URL: https://www.openssl.org/.

[107] Jingyu Pan, Fan Zhang, Kui Ren, and Shivam Bhasin. "One Fault is All it Needs: Breaking Higher-Order Masking with Persistent Fault Analysis". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2019, pp. 1–6. DOI: 10.23919/DATE.2019.8715260.

[108] Pancake. *Radare2*. 2006. URL: https://github.com/radareorg/radare2.

[109] David Pellerin. *FPGA Accelerated Computing Using AWS F1 Instances*. 2017. URL: https://old.hotchips.org/wp-content/uploads/hc%7B%5C_%7Darchives/hc29/HC29.22-Tuesday-Pub/HC29.22.50-FPGA-Pub/HC29.22.544-AWS-F1-Pellerin-Amazon%20v4.pdf.

[110] Gilles Piret and Jean-Jacques Quisquater. “A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad”. In: *Workshop on Crypto-graphic Hardware and Embedded Systems (CHES 2003)*. Vol. 2779. Sept. 2003, pp. 77–88. DOI: 10.1007/978-3-540-45238-6_7.

[111] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. “VoltJockey”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, Nov. 2019, pp. 195–209. DOI: 10.1145/3319535.3354201.

[112] J.-J. Quisquater and C Couvreur. “Fast decipherment algorithm for RSA public-key cryptosystem”. In: *Electronics Letters* 18.21 (1982), p. 905. DOI: 10.1049/el:19820617.

[113] Jean-Jacques Quisquater and David Samyde. “ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards”. In: *E-smart*. Vol. 2140. 2001, pp. 200–210. DOI: 10.1007/3-540-45418-7_17.

[114] Behzad Razavi. *Fundamentals of Microelectronics*. Ed. by NJ: Wiley Hoboken. 2008.

[115] Kaveh Razavi, Ben Gras, Erik Bosman, et al. “Flip Feng Shui: Hammering a Needle in the Software Stack”. In: *Proceedings of the 25th USENIX Conference on Security Symposium*. Austin, TX: USENIX Association, 2016, pp. 1–18. DOI: 10.5555/3241094.3241096.

[116] R. L. Rivest, A. Shamir, and L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems”. In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126. DOI: 10.1145/359340.359342.

[117] Joaquin Romo. *DDR Memories Comparison and overview*. Tech. rep. URL: https://www.nxp.com/docs/en/supporting-information/BeyondBits2article17.pdf.

[118] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O’Flynn. “IoT Goes Nuclear: Creating a Zigbee Chain Reaction”. In: *IEEE Security & Privacy* 16.1 (Jan. 2018), pp. 54–62. DOI: 10.1109/MSP.2018.1331033.

[119] D. Sanlyde, S. Skorobogatov, R. Anderson, and J.-J. Quisquater. “On a new way to read data from memory”. In: *First International IEEE Security in Storage Workshop, 2002. Proceedings.* IEEE Comput. Soc, 2003, pp. 65–69. DOI: 10.1109/SISW.2002.1183512.

[120] Stephan van Schaik, Alyssa Milburn, Sebastian Osterlund, et al. “RIDL: Rogue In-Flight Data Load”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019, pp. 88–105. DOI: 10.1109/SP.2019.00087.

[121] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. “Remote inter-chip power analysis side-channel attacks at board-level”. In: *Proceedings of the International Conference on Computer-Aided Design*. New York, NY, USA: ACM, Nov. 2018, pp. 1–7. DOI: 10.1145/3240765.3240841.

[122] Falk Schellenberg, Dennis R.E. Gnad, Amir Moradi, and Mehdi B. Tahoori. “An inside job: Remote power analysis attacks on FPGAs”. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Mar. 2018, pp. 1111–1116. DOI: 10.23919/DATE.2018.8342177.

[123] Mark Seaborn and Thomas Dullien. “Exploiting the DRAM rowhammer bug to gain kernel privileges”. In: *BlackHat 2015* March (2015). URL: https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

[124] Hwajeong Seo, Taehwan Park, Janghyun Ji, and Howon Kim. “Lightweight Fault Attack Resistance in Software Using Intra-instruction Redundancy, Revisited”. In: *Lecture Notes in Computer Science*. 2018, pp. 3–15. DOI: 10.1007/978-3-319-93563-8_1.

[125] Sergei P. Skorobogatov. “Semi-invasive attacks-a new approach to hardware security analysis”. PhD thesis. 2005, p. 144. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf.

[126] Sergei P. Skorobogatov and Ross J. Anderson. “Optical Fault Induction Attacks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 2523. 2003, pp. 2–12. DOI: 10.1007/3-540-36400-5_2.

[127] Sung-Ming Yen and Marc Joye. “Checking before output may not be enough against fault-based cryptanalysis”. In: *IEEE Transactions on Computers* 49.9 (2000), pp. 967–970. DOI: 10.1109/12.869328.

[128] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. “CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management”. In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017. DOI: 10.5555/3241189.3241272.

[129] Sébastien Tiran, Sébastien Ordas, Yannick Teglia, et al. “A model of the leakage in the frequency domain and its application to CPA and DPA”. In: *Journal of Cryptographic Engineering* 4.3 (Sept. 2014), pp. 197–212. DOI: 10.1007/s13389-014-0074-x.

[130] Linus Torvalds. *Linux Kernel*. 1991. URL: https://github.com/torvalds/linux.

[131] Steve Trimberger and Steve McNeil. “Security of FPGAs in data centers”. In: *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*. IEEE, July 2017, pp. 117–122. DOI: 10.1109/IVSW.2017.8031556.

[132] Miho Ueno, Masanori Hashimoto, and Takao Onoye. “Real-time on-chip supply voltage sensor and its application to trace-based timing error localization”. In: *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*. IEEE, July 2015, pp. 188–193. DOI: 10.1109/IOLTS.2015.7229857.

[133] Jo Van Bulck, Marina Minkin, Ofir Weisse, et al. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient out-of-Order Execution”. In: *Proceedings of the 27th USENIX Conference on Security Symposium*. SEC’18. USA: USENIX Association, 2018. DOI: 10.5555/3277203.3277277.

[134] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, et al. “LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection”. In: *2020 IEEE Symposium on Security and Privacy (SP)*. Vol. 2020-May. IEEE, May 2020, pp. 54–72. DOI: 10.1109/SP40000.2020.00089.

[135] Jo Van Bulck, Frank Piessens, and Raoul Strackx. “SGX-Step”. In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. SysTEX’17. New York, NY, USA: ACM, Oct. 2017, pp. 1–6. DOI: 10.1145/3152701.3152706.

[136] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, et al. “Drammer”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. New York, NY, USA: ACM, Oct. 2016, pp. 1675–1689. DOI: 10.1145/2976749.2978406.

[137] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, et al. “GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10885 LNCS. 2018, pp. 92–113. DOI: 10.1007/978-3-319-93411-2_5.

[138] S. J. Wind, J. Appenzeller, R. Martel, et al. *More than Moore*. Ed. by Guo Qi Zhang and Alfred Roosmalen. Vol. 3. 2. Boston, MA: Springer US, 2009, pp. 758–62. DOI: 10.1007/978-0-387-75593-9.

[139] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. “One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation”. In: *Proceedings of the 25th USENIX Conference on Security Symposium*. Austin, TX: USENIX Association, 2016, pp. 19–35. DOI: 10.5555/3241094.3241097.

[140] Xilinx Inc. *[UG474] 7 Series FPGAs Configurable Logic Block*. Tech. rep. 2016, pp. 1–72. URL: https://www.xilinx.com/support/documentation/user%7B%5C_%7Dguides/ug474%7B%5C_%7D7Series%7B%5C_%7DCLB.pdf.

[141] Xilinx Inc. *[UG585] Zynq-7000 AP SoC Technical Reference Manual*. Tech. rep. 2012, pp. 1–1825. URL: https://www.xilinx.com/support/documentation/user%7B%5C_%7Dguides/ug585-Zynq-7000-TRM.pdf.

[142] Asier Goikoetxea Yanci, Stephen Pickles, and Tughrul Arslan. “Characterization of a Voltage Glitch Attack Detector for Secure Devices”. In: *2009 Symposium on Bio-inspired Learning and Intelligent Systems for Security*. IEEE, Aug. 2009, pp. 91–96. DOI: 10.1109/BLISS.2009.18.

[143] Asier Goikoetxea Yanci, Stephen Pickles, and Tughrul Arslan. “Detecting Voltage Glitch Attacks on Secure Devices”. In: *2008 Bio-inspired, Learning and Intelligent Systems for Security*. IEEE, Aug. 2008, pp. 75–80. DOI: 10.1109/BLISS.2008.26.

[144] Yuval Yarom and Katrina Falkner. “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. San Diego, CA: USENIX Association, 2014, pp. 719–732. DOI: 10.5555/2671225.2671271.

[145] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, et al. “Persistent Fault Analysis on Block Ciphers”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.3 (2018), pp. 150–172. DOI: 10.13154/tches.v2018.i3.150-172.

[146] Fan Zhang, Yiran Zhang, Huilong Jiang, et al. “Persistent Fault Attack in Practice”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.2 (2020), pp. 172–195. DOI: 10.13154/tches.v2020.i2.172-195.

[147] Lu Zhang, Luis Vega Gutierrez, and Michael Bedford Taylor. “Power Side Channels in Security ICs: Hardware Countermeasures”. In: *arXiv* (2016). URL: http://arxiv.org/abs/1605.00681.

[148] Mark Zhao and G. Edward Suh. “FPGA-Based Remote Power Side-Channel Attacks”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2018, pp. 229–244. DOI: 10.1109/SP.2018.00049.

[149] Yongbin Zhou and DengGuo Feng. “Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing.” In: *IACR Cryptology ePrint Archive* 2005 (2005), p. 388. URL: https://www.iacr.org/cryptodb/data/paper.php?pubkey=12722.

[150] Kenneth M. Zick and John P. Hayes. “Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems”. In: *ACM Transactions on Reconfigurable Technology and Systems* 5.1 (Mar. 2012), pp. 1–26. DOI: 10.1145/2133352.2133353.

[151] Kenneth M. Zick, Meeta Srivastav, Wei Zhang, and Matthew French. “Sensing nanosecond-scale voltage attacks and natural transients in FPGAs”. In: *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA ’13*. New York, New York, USA: ACM Press, 2013, p. 101. DOI: 10.1145/2435264.2435283.

[152] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Bruno Robisson. “Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter”. In: *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, May 2014, pp. 130–135. DOI: 10.1109/HST.2014.6855583.

[153] Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Assia Tria. “Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism”. In: *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. IEEE, July 2013, pp. 110–115. DOI: 10.1109/IOLTS.2013.6604060.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *2021LYSEM034*

Joseph GRAVELLIER

Remote Hardware Attacks on Connected Devices

Speciality : Microelectronic

Keywords : hardware attacks, remote attacks, on-chip sensing, SoC, FPGA, power side-channel, fault injection, delay-lines, SideLine, FaultLine.

Abstract : In this thesis we evaluate software-based hardware attacks, a novel attack family that targets connected devices such as IoT products, smartphones or cloud datacenters. These attacks take advantage of hardware resources that can be directly accessed from software in complex integrated circuits and use them to conduct fault injection and side-channel analysis exploits. Because they are triggered by software code, they can be launched remotely and on multiple victim devices regardless of their physical location. Through the evaluation of software-based hardware attacks we aim at assessing the level of threat they pose to connected devices security. The challenge is considerable since any connected device is potentially endangered.

Through our experiments, we discovered generic attack vectors widely implemented in recent SoCs that could enable remote hardware attacks. We conducted FPGA-to-FPGA and FPGA-to-CPU attacks and demonstrated that remote power analysis was feasible. These new attack paths represent a serious threat for FPGA-based systems especially when applied to cloud FPGAs and heterogeneous SoCs. Then, we went further by proving that delay-line components, widely implemented in high-end SoCs, could be used for conducting fault injection and side-channel attacks. We built various scenarios such as CPU-to-MCU attacks on complex operating systems and demonstrated that software-based hardware attacks could successfully break the logical isolation between processes and lead to the extraction of sensitive information. Our research works describe the methodology adopted to build and conduct the attacks along with countermeasure proposals to mitigate their impact.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *2021LYSEM034*

Joseph GRAVELLIER

Attaques Matérielles à Distance des Objets Connectés

Spécialité: Microélectronique

Mots clefs : attaques matérielles, attaques à distance, capteurs intégrés, SoC, FPGA, analyse de courant, injection de faute, SideLine, FaultLine.

Résumé : Depuis les années 90 et la découverte de vulnérabilités physiques dans les circuits intégrés, l'étude de la résistance aux attaques matérielles est devenue un maillon indispensable du développement de produits sécurisés (cartes à puces, modules de chiffrement, microcontrôleurs). Une attaque matérielle vise à extraire les données secrètes contenues dans un circuit intégré. Les techniques généralement employées à cet usage analysent les émanations physiques (attaques par canaux-cachés) ou perturbent le fonctionnement (attaques par injection de faute) d'un appareil ciblé. À la différence des attaques logicielles qui peuvent être menées à distance à travers un réseau, il est communément admis que les attaques matérielles sont locales car elles nécessitent l'utilisation d'équipements de laboratoire. Cependant, une multitude de travaux de recherche récents démontrent que ce postulat n'est plus valable. En effet, avec l'adoption massive des objets connectés, la complexification des circuits intégrés et l'apparition de systèmes multi-utilisateurs, il devient possible de mener des attaques matérielles à distance. C'est-à-dire, sans accès direct au circuit ciblé.
Cette thèse propose d'étudier ces nouveaux chemins d'attaque qui exploitent des vulnérabilités physiques à distance. Plus précisément elle s'intéresse à celles qui utilisent du logiciel comme vecteur d'attaque matérielle. Il peut s'agir par exemple d'un programme malveillant envoyé à des dizaines, voire des milliers d'appareils connectés. Une fois actif, il identifie des ressources matérielles présentes dans les systèmes ciblés (capteurs, régulateurs) et les détourne de leur rôle initial afin de mener des attaques matérielles. À l'instar d'attaques reconnues telles que Rowhammer, CLKSCREW ou Platypus, les travaux réalisés durant ces trois années de recherche contribuent à mettre en avant le danger potentiel des attaques matérielles à distance. Cette thèse contient des résultats variés allant de l'analyse de consommation sur FPGA à de l'injection de faute sur processeurs complexes. Elle décrit la menace potentielle des attaques matérielles à distance, notamment au regard de l'adoption d'entités de sécurité intégrées dans les processeurs complexes et de l'accroissement des services connectés en général (IoT, Cloud). Toutes ces études ont été menées de façon à être reproductibles et réalisables à distance. Elles visent à préparer les différents acteurs de la sécurisation des objets connectés à cette menace naissante et ainsi éviter sa mise en exécution future sur des milliers d'appareils à travers le monde.