



## Agence Nationale de la Sécurité des Systèmes d'Information

### Université GRENOBLE ALPES

Habilitation à diriger les recherches présentée par **Guillaume BOUFFARD**

Soutenue le **23 septembre 2025**

Discipline **Informatique et mathématiques appliquées**

# Contributions à la sécurité des logiciels embarqués dans la chaîne de confiance

#### Composition du jury

<i>Rapporteurs</i>	Jean-Max DUTERTRE	Professeur, École des Mines de SAINT-ÉTIENNE
	Aurélien FRANCILLON	Professeur, EURECOM
	Patrick SCHAUMONT	Full professor, WORCESTER Polytechnic Institute
<i>Examineurs</i>	Jessy CLÉDIÈRE	Directeur de recherche, CEA Centre de GRENOBLE
	Karine HEYDEMANN	Senior scientist, THALES CDI et chercheuse HDR associée au LIP6/SORBONNE Université
	Marie-Laure POTET	Professeure des Universités, GRENOBLE INP - Université GRENOBLE ALPES
	Sébastien VARRETTE	Responsable de laboratoire, Agence Nationale de la Sécurité des Systèmes d'Information

## COLOPHON

Mémoire de thèse intitulé « Contributions à la sécurité des logiciels embarqués dans la chaîne de confiance », écrit par Guillaume BOUFFARD, achevé le 3 septembre 2025, composé au moyen du système de préparation de document [L<sup>A</sup>T<sub>E</sub>X](#) et de la classe [yathesis](#) dédiée aux thèses préparées en France.

# Remerciements

Au milieu de l'hiver, j'apprenais enfin qu'il y avait en moi un été invincible.

---

Albert CAMUS. *L'Été*. 1954

On dit souvent que le plus important dans un voyage n'est pas la destination, mais le chemin parcouru. Ou, pour reprendre les mots d'ORELSAN, « *c'qui compte c'est pas l'arrivée, c'est la quête* »<sup>1</sup>.

Cette Habilitation à Diriger les Recherches (HDR) n'est pas seulement un point d'aboutissement. Elle est aussi la trace d'années de travail, de découvertes, d'essais et d'erreurs, mais surtout de rencontres. Sur cette route, il y a eu des guides bienveillants, des compagnons de route fidèles, des éclaireurs qui ont ouvert de nouvelles perspectives et des alliés qui ont tendu la main dans les moments d'incertitude. C'est à vous que je veux adresser ces lignes, pour vous dire que chacun de vos gestes, conseils, critiques ou encouragements a compté. Ce manuscrit porte vos empreintes autant que les miennes.

Je tiens à remercier les membres de mon jury pour avoir accepté de consacrer leur temps et leur expertise à l'évaluation de ce travail. Merci à Jean-Max DUTERTRE, Aurélien FRANCILLON et Patrick SCHAU MONT d'avoir accepté de rapporter cette HDR, ainsi qu'à Jessy CLÉDIÈRE, Karine HEYDEMANN, Marie-Laure POTET et Sébastien VARRETTE d'avoir accepté d'en examiner le contenu et d'enrichir la réflexion par leurs retours. Je veux adresser à Karine et Marie-Laure une reconnaissance particulière : leur soutien et leurs encouragements, dans les moments décisifs, m'ont donné l'élan nécessaire pour mener ce travail jusqu'à son terme. Je tiens également à remercier Paolo MAISTRI pour son soutien à l'organisation de ma soutenance.

I would like to express my sincere gratitude to Patrick Schaumont for his time, insightful feedback, and valuable perspective on my HDR work. His expertise and thoughtful comments have greatly contributed to the quality and depth of this manuscript.

Dans cette quête, certaines rencontres changent définitivement la route. Pour moi, Jean-Louis LANET en fait partie. C'est lui qui, alors que j'étais en master, m'a ouvert les portes de la recherche en sécurité informatique, me donnant l'envie d'explorer ce domaine encore inconnu. Il a ensuite accepté de diriger ma thèse de doctorat, m'accompagnant avec exigence, bienveillance et passion. Au fil des années, il est devenu bien plus qu'un directeur de thèse : un mentor et un ami, présent dans les moments décisifs comme dans les détours imprévus. Bien qu'il ait quitté depuis longtemps les routes académiques, son empreinte demeure dans chacune de mes étapes, comme ces balises que l'on croise au détour d'un chemin et qui rappellent la voie à suivre.

Mon arrivée, en 2014, à l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) a

---

1. ORELSAN, « La Quête », *Civilisation*, Warner Chappell Music France, nov. 2021.

marqué le début d'un nouveau chapitre dans cette quête. Dès mes premiers pas, j'ai eu la chance d'y trouver le soutien d'Eliane JAULMES, responsable du Laboratoire Sécurité des Composants (LSC), puis de Sébastien VARRETTE accompagné d'Alain OZANNE, son adjoint, en charge du Laboratoire Architectures Matérielles et Logicielles (LAM). Tous trois ont su tempérer parfois mes élans et m'ouvrir des chemins que je n'aurais pas explorés seul.

Je souhaite également remercier les différentes strates managériales de l'ANSSI : José ARAUJO, Eric SALIBA, Aline GOUGET, Renaud LABELLE, Annaïg ANDRO et Vincent STRUBEL, dont les interrogations et les échanges, parfois en marge de mes travaux, m'aidant à affiner ma direction et à faire mûrir mes réflexions.

À l'ANSSI, j'ai eu la chance de croiser de nombreux compagnons de route qui, chacun à leur manière, ont guidé mes réflexions, éclairé mes doutes ou enrichi mes idées. Je souhaite remercier tout particulièrement les membres que j'ai croisés au LSC : Karim KHALFALLAH, Romain POUSSIER et Franck RONDEPIERRE

En janvier 2023, ma quête m'a conduit jusqu'au LAM. Après un long chemin, j'y ai trouvé une oasis : un lieu où la bienveillance et la confiance de chacun ont offert l'espace et la sérénité nécessaires pour travailler à cette HDR. Je souhaite remercier chaleureusement Olivier BAL-PÉTRÉ, Luc BONNAFOUX, Nicolas BOUCHINET, François JOLIVET, Gabriel KERNEIS, Serge LOTTIAU, Arnauld MICHELIZZA, Stéphane NEVEU, Alain OZANNE, Philippe TRÉBUCHET et Sébastien VARRETTE, dont le soutien, les échanges et les encouragements ont été comme une eau claire pour nourrir mes réflexions et faire mûrir mes idées.

Je tiens également à remercier l'ensemble de l'équipe du Centre de Certification National, et plus particulièrement Géraldine AVOUÉ, Soline RENNER et Jérôme VIDAL, dont la disponibilité et la qualité des échanges ont été d'une aide précieuse sur mon chemin. Je n'oublie pas Valentin HOUCOUAS, Louisa KHATI, José LOPES ESTEVES, Alexandre MAGLOIRE, Hugo MANIA, Ange MARTINELLI et Philippe VALEMOIS, compagnons de route chers à mon cœur, dont l'amitié demeure solide malgré les trop rares moments partagés.

Je veux aussi remercier Coralie CHABOT, Arnaud MINART et Anne PROTAS dont l'écoute et la disponibilité ont été des appuis précieux. Ils font partie de ces présences discrètes qui, par de simples gestes, rendent les journées plus légères et permettent au voyage de se poursuivre avec sérénité.

Je souhaite également remercier celles et ceux qui ont embarqué avec moi dans les travaux présentés dans cette HDR. Mes doctorants, qu'ils aient déjà atteint le port ou qu'ils soient encore en pleine traversée, ont été des équipiers précieux et sont devenus des amis. À ceux qui ont déjà mené leur navire à bon port, Amélie MAROTTA, Vincent GIRAUD et Thomas TROUCHKINE, je garde en mémoire nos traversées parfois agitées mais toujours porteuses de découvertes. À ceux qui voguent encore sur cette mer d'incertitudes et de promesses, Jonah ALLE MONNE et Gwenn LE GONIDEC, je me réjouis des horizons que nous allons encore explorer ensemble.

Aucune expédition ne se mène seul, et si les doctorants sont les équipages en mer, les co-encadrants sont les capitaines restés à terre, veillant à ce que le cap soit maintenu et que chaque traversée se déroule au mieux. Les thèses que j'ai dirigées ou co-dirigé encore doivent beaucoup à ces partenaires de route, dont l'engagement et la vigilance ont été déterminants à chaque étape. Je tiens à exprimer ma profonde gratitude à Jessy CLÉDIÈRE, Damien COUROUSSÉ, Rachid DAFALI, Mathieu JAN, Maria MÉNDEZ REAL, David NACCACHE, Jean-Christophe PRÉVOTET et Olivier SENTIEYS. Leur expertise, leur disponibilité et leur implication constante sont des phares dans cette aventure scientifique, éclairant la voie et rendant chaque arrivée possible.

Et puis, il y a celles et ceux qui, dans les coulisses, ont apporté leur savoir-faire et leur énergie à certains travaux présentés dans cette HDR. Comme des artisans du navire qui, parfois le temps d'une escale, contribuent à renforcer la coque, ajuster les voiles ou affiner les instruments, ils ont laissé leur marque sur ce voyage scientifique. Je tiens à remercier chaleureusement Ever ATILANO ROSALES, Yanis BELKHEYAR, Angie-Sofia BIKOU, Léo GASPARD, Louiza MALKI et Guillaume P., pour leur travail, leur impli-

cation et leur enthousiasme.

Au détour d'un sentier de cette quête, certaines rencontres se sont révélées précieuses et durables. Ces compagnons de route, d'abord croisés dans le cadre d'une collaboration, sont devenus des amis avec qui j'ai partagé bien plus que des objectifs scientifiques. Je tiens à remercier Yoann CHERGUI, Laura COMAN, David EL BAZE, Hélène LE BOUDER, Julien EYNARD, Patrick HADDAD, Nicolas HUGGET, Ambre IOOSS, Ronan LASHERMES, Boris SIMUNOVIC, Adrian THILLARD Victor LOMNÉ et Besma ZEDDINI pour les discussions passionnées, les instants complices et l'amitié qui perdure bien au-delà de nos projets communs.

Je n'oublie pas celles et ceux qui ont pris le temps de relire ce manuscrit pour l'améliorer et le rendre aussi clair et précis que possible. Leur regard attentif a permis de gommer bien des aspérités et de renforcer la cohérence de l'ensemble.

Enfin, je veux remercier mes amis Mohamed Amine BOUZZOUNI, Jean DUBREUIL, Tom KHEFIF, Damien MARION et Tiana RAZAFINDRALAMBO, qui ont toujours cru en moi et m'ont rappelé, à chaque étape de ce voyage, que la confiance et l'amitié sont des forces aussi précieuses que le savoir.

Aucune quête ne se mène sans un port d'attache solide. Je souhaite remercier ma sœur Jennifer et son conjoint Cyril, pour leur présence et leur soutien discret mais constant. Je remercie également mes parents, qui m'ont donné le goût de la curiosité et la force d'aller au bout de mes projets. Et, plus que tout, je veux exprimer toute ma gratitude à Anne, ma merveilleuse femme, pour sa patience, son écoute et son indéfectible soutien dans les moments calmes comme dans les tempêtes. Sans vous, cette aventure n'aurait pas eu la même saveur.

Si cette HDR marque une étape, elle reste avant tout une invitation à poursuivre la quête, ensemble.



# Acronyms

A | B | C | D | E | F | G | I | J | L | M | O | P | R | S | T | V

## A

**ADAS** Advanced Driver Assistance Systems 60  
**APDU** Application Protocol Data Unit 22–24  
**API** Application Programming Interface 18, 19, 21, 24, 31, 109  
**ASIC** Application-Specific Integrated Circuit 34, 38, 39, 58, 89

## B

**BCV** Byte Code Verifier 15–20, 52

## C

**CAN** Controller Area Network 61, 62  
**CFI** Control Flow Integrity 32, 46, 49, 50  
**CFSSI** Centre de Formation à la Sécurité des Systèmes d’Information 87, 88  
**CHERI** Capability Hardware Enhanced RISC Instructions 32, 46, 49  
**CoT** Chain of Trust 1, 3–7, 11, 22, 29, 43, 51–54, 56, 61–63, 88  
**CPU** Central Processing Unit 2, 5–9, 12, 13, 26–36, 38, 40, 41, 44, 49, 51–59, 110

## D

**DRM** Digital Rights Management 3, 6, 9, 89

## E

**EMFI** Electromagnetic Fault Injection 38, 90  
**ENS** Ecole Normale Supérieure 85, 86, 91, 93

## F

**FPGA** Field-Programmable Gate Array 34, 38, 39, 89, 90, 93

## G

**GPU** Graphics Processing Unit 31, 32, 49

## I

**IoT** Internet of Things 14

**ISA** Instruction Set Architecture 26, 33, 35–37, 39, 41, 53, 54

**ITSEF** Information Technology Security Evaluation Facility 1, 12–14, 23, 52, 86, 88

## J

**JCDK** Java Card Development Kit 17–19

**JCVM** Java Card Virtual Machine 1, 7, 12, 14–21, 23–25, 52, 90, 109

**JVM** Java Virtual Machine 14, 16, 109

## L

**LAM** Hardware and Software Architectures Lab 1, 51, 85, 92

**LSC** Component Security Lab 1, 51, 85, 93

## M

**MMU** Memory Management Unit 32, 36, 44, 49, 50

**MPU** Memory Protection Unit 21

## O

**OS** operating system i, 4, 12–15, 17, 19–21, 24, 25, 29–31, 37, 43–46, 48, 50, 55

**OTP** One-Time Password 2, 12

## P

**PAC** Pointer Authentication Code 32, 46, 49, 50

**PIM** Platform Integrity Module 2, 3

**PMU** Power Management Unit 40, 41, 49, 53

## R

**R&D** Research & Development 1

**REE** Rich Execution Environment 4–7, 9, 10, 29–32, 40, 41, 43–56, 59, 61–63, 88, 89

**RoT** Root of Trust 1–9, 11, 12, 22, 26, 28–30, 41, 43–45, 51, 52, 54, 56–59, 61–63, 88, 89

**RTL** Register Transfer Level 34, 38, 39, 58

## S

**SE** Secure Element 2, 5, 7, 11–18, 20, 22–31, 40, 47, 51–53, 57–59, 61, 63, 88, 109

**SGX** Software Guard eXtensions 4

**SoC** System-on-Chip 2, 7, 9, 12, 29–33, 36, 40, 41, 49, 53, 56, 57, 59, 63, 90, 92

## T

**TEE** Trusted Execution Environment 4–9, 11, 28–32, 40, 41, 43–46, 48–56, 58, 59, 61–63, 88–91

**TPM** Trusted Platform Module 2, 3

## V

**VPU** Video Processing Unit 31, 32, 49

# Contents

<b>Remerciements</b>	<b>iii</b>
<b>Acronyms</b>	<b>vii</b>
<b>Contents</b>	<b>i</b>
<b>List of Figures</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Research Context . . . . .	5
1.3 Challenges . . . . .	5
1.4 My Contributions . . . . .	6
1.5 Organization of this Manuscript . . . . .	9
<b>2 Contributions to the Hardware Root of Trust Security</b>	<b>11</b>
2.1 Common Secure Element Architecture . . . . .	12
2.2 Security of the On-Chip Operating System Layout . . . . .	14
2.3 Security Analysis of the Input/Output Interfaces . . . . .	22
2.4 Analysis of CPU Security Against Hardware Attacks . . . . .	26
2.5 Conclusion and Perspectives . . . . .	28
<b>3 Contributions to the Trusted Execution Environment Security</b>	<b>29</b>
3.1 Common Trusted Execution Environment Architecture . . . . .	30
3.2 Hardware Architecture of Application Processors . . . . .	31
3.3 Impact of Hardware Attacks on High-Performance Processors . . . . .	33
3.4 Impact of Environment on Application Processors Security . . . . .	40
3.5 Conclusion and Perspectives . . . . .	41
<b>4 Contributions to Execute Sensitive Applications in the Rich Execution Environment</b>	<b>43</b>
4.1 Common Rich Execution Environment Architecture . . . . .	44
4.2 Software Security in the Rich Execution Environment . . . . .	45
4.3 Rich Execution Environment Hardware Security Challenges . . . . .	49
4.4 Conclusion and Perspectives . . . . .	50

---

<b>5 Conclusion and Perspectives</b>	<b>51</b>
5.1 Summary of Activities Introduced in this Manuscript . . . . .	52
5.2 Perspectives . . . . .	54
5.3 Conclusion . . . . .	63
<b>Bibliography</b>	<b>65</b>
<b>A Curriculum vitæ</b>	<b>85</b>
A.1 Administrative Information . . . . .	85
A.2 Professional Experience and Degrees . . . . .	85
A.3 Teaching . . . . .	87
A.4 Research Activities at ANSSI . . . . .	87
A.5 Scientific Responsibilities . . . . .	91
A.6 Full List of My Publications . . . . .	96
A.7 Ph.D. Thesis Reports and Defense Minutes . . . . .	101
<b>Glossary</b>	<b>109</b>

# List of Figures

1.1	Different use cases of hardware Root of Trust . . . . .	3
1.2	Classic Chain of Trust architecture based on hardware Root of Trust. . . . .	4
1.3	Overview of my research activities . . . . .	7
2.1	Overview of the Secure Element hardware and software architecture . . . . .	12
2.2	Overview security model of the Java Card technology. . . . .	15
2.3	Java Card method resolution process . . . . .	18
2.4	Ambiguity in Method Resolution . . . . .	19
2.5	ISO/IEC 7816 protocol layers . . . . .	23
2.6	Overview of ISO/IEC 7816 stack architecture embedded in a Secure Element upon each software level . . . . .	24
2.7	Fuzzing architecture of ISO/IEC 7816 protocol. . . . .	25
3.1	Common Trusted Execution Environment software architecture . . . . .	31
3.2	Overview of application System-on-Chip hardware and software architecture . . . . .	32
3.3	Fault propagation through digital device abstraction layers with examples of fault effects and characterization levels . . . . .	34
3.4	Top-down approach to characterize fault effects on in-order application processors micro-architecture blocks. . . . .	35
4.1	A common Rich Execution Environment architecture . . . . .	45



# Chapter 1

## Introduction

This habilitation to supervise research thesis provides an overview of my research activities at the [National Cybersecurity Agency of France \(ANSSI\)](#) Expertise department. After earning my Ph.D. in October 2014, which focused on analyzing the security of [Java Card Virtual Machine \(JCVM\)](#) implementations embedded in hardware [Roots of Trust \(RoTs\)](#) against both software and hardware attacks, I joined the [ANSSI](#) in November 2014. There, I became expert in the [ANSSI Research & Development \(R&D\)](#) labs, initially in the [Component Security Lab \(LSC\)](#) and later in the [Hardware and Software Architectures Lab \(LAM\)](#).

I have expanded my research to encompass the security of embedded software implementations deployed in each element of the [Chain of Trust \(CoT\)](#), addressing threats from both software and hardware attacks. While the literature mainly focus on the security of cryptographic implementations, the security of embedded software has been less studied. My research aims to address this gap and to anticipate the risks targeting each element of a [CoT](#). In particular, we have analyzed several implementations and discovered vulnerabilities in both their hardware [[Tro+21](#)] and software [[DB21](#)]. We also proposed several countermeasures to fix them. For example, in the field of hardware [RoT](#), my contributions have significantly enhanced the robustness of security products [[LB15](#)]. This has encouraged developers to strengthen their solutions, [Information Technology Security Evaluation Facilitys \(ITSEFs\)](#) to broaden their criteria [[DB21](#); [ldr+17](#)] and end-users to benefit from greater security.

### 1.1 Background

In our daily lives, we frequently perform sensitive operations, such as online banking, secure communication, or accessing private information. These activities require a secure host to protect sensitive data against unauthorized access or tampering. This host must provide a trusted environment, ensuring that information cannot be extracted and security functions remain uncompromised. To achieve this level of trust, such an element serves as a foundational component known as a [RoT](#). A [RoT](#) acts as the cornerstone for building a secure environment where sensitive functions are executed.

One of the most familiar examples is the **Secure Element (SE)**, commonly seen in smart cards used for payments or identity verification. Today, **RoTs** are prevalent, embedded in a wide range of devices such as computers, vehicles, TVs, video game consoles, and smartphones.

### 1.1.1 The Root of Trust

The **RoT** is defined by GlobalPlatform [Glo18b] as a combination of a computing engine, code, and potentially associated data co-located on the same platform, designed to provide foundational security services. These components are implicitly trusted as the base of a platform's security architecture, meaning their integrity and behavior cannot be verified by any preceding component.

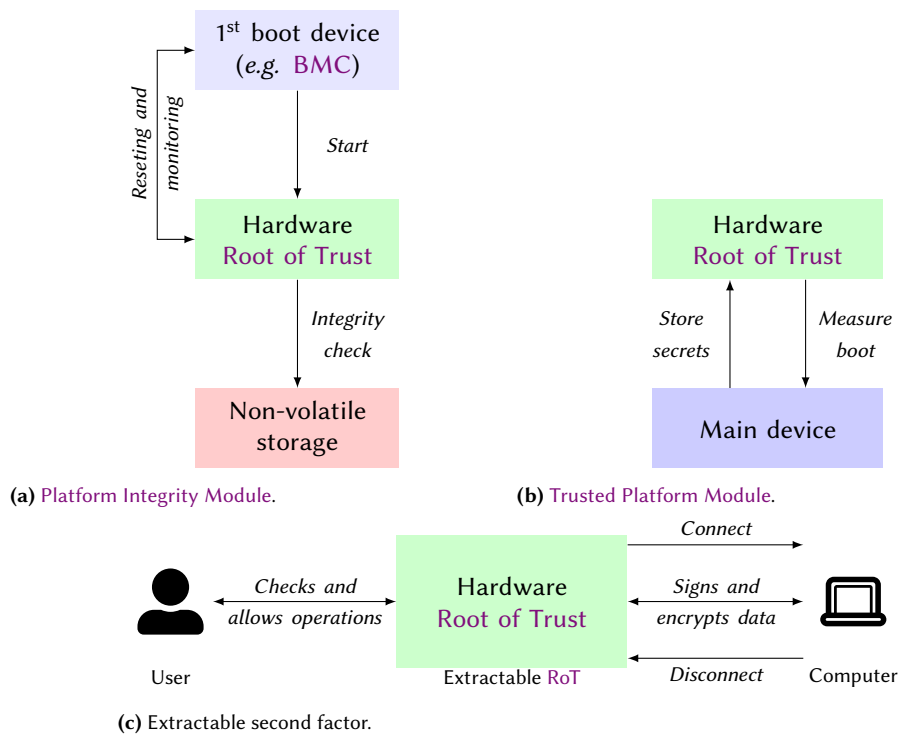
For the purpose of this manuscript, I have intentionally defined the **RoT** as the minimal component that is accessible and suitable to meet the needs of **ANSSI** beneficiaries. This level of granularity ensures that the research presented here remains relevant and applicable. It is crucial that the elements studied are representative of components commonly available on the market.

Critical security functions in an information system are built upon the **RoT**. It serves as the foundation for establishing trust within the system, ensuring that security operations, such as authentication and integrity verification, are carried out reliably and securely. To guarantee this trust, the **RoT** must be inherently trustworthy, which is often verified through a high-level security evaluation performed by a third party. Such assessments, typically conducted under the Common Criteria security evaluation scheme [Eur14; Eur22], demonstrate the **RoT**'s resilience against highly sophisticated attackers. Additionally, an attestation system [Adv23; Int21] is generally employed to assert the authenticity of the **RoT** and validate the platform version.

To ensure comprehensive security, a system must be secure from the hardware layer to the software layer. **RoT** is often implemented as a hardware secure module [Ram23]. This module can be a silicon-based hardware component embedded into a **System-on-Chip (SoC)** or a discrete component on the same motherboard as the system's main **Central Processing Unit (CPU)**.

A hardware **RoT** supports key use cases [Ope23], such as verifying system integrity, performing a measured boot as part of a secure boot process, and serving as an extractable second factor for authentication, as introduced in **Figure 1.1**. Firstly, verifying system integrity means making sure that the system's hardware and software have not been tampered with, thus ensuring the trustworthiness of the computing environment. In this context, the **RoT** is known as **Platform Integrity Module (PIM)**, as illustrated in **Figure 1.1a**. Secondly, performing a measured boot as part of a secure boot process entails measuring and recording the state of the system components during the boot sequence, which helps in detecting and preventing unauthorized modifications. For this use case, the hardware **RoT** is referred to as **Trusted Platform Module (TPM)**, as introduced in **Figure 1.1b**. Lastly, serving as an extractable second factor for authentication provides an additional layer of security by requiring a physical token or device, in addition to a password to authenticate users, thereby enhancing protection against unauthorized access, as shown in **Figure 1.1c**. Such devices can also be used to securely store secrets like **One-Time Passwords (OTPs)**, SSH or GPG keys, and even passwords, providing a secure enclave for sensitive data

and further strengthening the overall security.

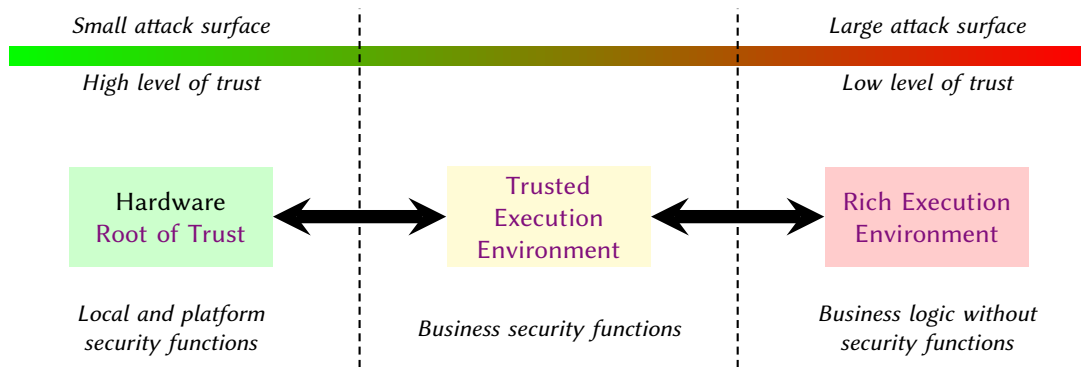


**Figure 1.1** – Different use cases of hardware RoT, inspired by [Ope23].

To minimize the attack surface, current implementations of hardware RoTs deliberately limit embedded features, often by restricting the device to execute only one application at a time to ensure secure operations. This approach significantly reduces performance and functionality. Such hardware RoTs alone are insufficient to meet the demands of modern applications, like streaming Digital Rights Management (DRM)-protected videos or encrypting data for cloud storage. To effectively balance high performance with robust security, it is essential to deploy a CoT that integrates multiple layers of RoTs, allowing for a more flexible and comprehensive security architecture.

### 1.1.2 The Chain of Trust

A CoT [Glo19] is a sequence of trust relationships that originates from a RoT and extends through various levels of security elements within a performance-oriented system. Each element in the chain verifies the next component, insuring the integrity and trustworthiness of the entire system. This hierarchical approach allows for a scalable and secure architecture, where multiple layers of hardware and software security can be implemented, each offering different levels of assurance and protection. The CoT is crucial in complex systems where diverse security requirements must be addressed. Figure 1.2 provides an overview of a modern, high-performance runtime environment tailored for mainstream consumer applications, based on a multi-stage CoT.



**Figure 1.2** – Classic CoT architecture based on hardware RoT. This architecture is generally implemented as-is in smartphones, laptops and cloud servers.

Each element that composes a CoT is designed to provide specific security features for the entire system, determined by the level of trust placed in the element and its security assurance level. A CoT can be divided into the following components:

**Hardware RoT** The hardware RoT, as introduced in Section 1.1.1, is the cornerstone of the security of the entire system. With a small attack surface and high security level, hardware RoTs provide *local and platform security functions*, such as secure boot [ANS23; App24] and platform key management [Int15].

**Trusted Execution Environment (TEE)** TEEs are initially designed to emulate the functionality of hardware RoTs, focusing on performance and resistance to tampering from software attacks [AF04]. Modern TEEs implementations such as AMD Security Processor [AMD23], Arm TrustZone [Arm17], and Intel Software Guard eXtensions (SGX)<sup>1</sup> [Int22a] claim to be resistant to software attacks. However, there are documented cases where vulnerabilities in these TEE implementations have been exploited through sophisticated software attacks [Cer+20; Sch+24]. Additionally, the TEE protection profile [Glo20b] has evolved to address resistance to some hardware attacks, recognizing the increasing sophistication of threat models [Jac+23; TSS17]. Despite these developments, no widely deployed TEE implementation have yet demonstrated resistance to a many of hardware attacks.

TEEs are primarily designed to run *business security functions* for applications running in the Rich Execution Environment (REE).

**Rich Execution Environment (REE)** The REE represents the final layer in the CoT, where high-performance applications operate. These applications are typically executed on modern *operating systems (OSes)* such as Linux or Windows for computers and servers, and Android or iOS for smartphones.

Modern OSes employ hardware-based security mechanisms, including memory protections and

1. Intel SGX has been deprecated on 11<sup>th</sup> and 12<sup>th</sup> generation Intel Core processors since 2021 [Tou22], although its development continues for Intel Xeon processors targeting cloud and enterprise applications [Rao22].

control-flow integrity, provided by the system CPU, to safeguard running software. However, the complexity of the runtime environment often makes comprehensive security assurance challenging. Applications in the REE generally focus on *business logic without security functions*, with security responsibilities commonly offloaded to TEEs or hardware RoTs via the TEE. Nevertheless, when developers of high-performance applications lack access to TEE or hardware RoT, they are compelled to implement security mechanisms within the REE itself. This issue is further explored in Section 1.3.

## 1.2 Research Context

In late 2014, SEs, such as smart cards, were the most widely used security components for executing sensitive operations. These components serve as hardware RoTs, ensuring local and platform security functions.

However, towards the end of the 2010s, there was a significant shift in the landscape of secure computing. Sensitive operations increasingly started being executed outside of traditional SEs, within the main system's CPU, inside TEEs. This shift was driven by the growing complexity and performance demands of applications. As a result, there was a need to develop new architectures that extended trust beyond the boundaries of SEs, leading to the adoption of systems based on a CoT.

By the early 2020s, the nature of security threats had evolved. Initially, attacks that were traditionally targeted at SEs began to be adapted and applied to TEEs [Vas+20; YSW18]. TEEs, designed to provide a secure and high-performance area within the main processor for executing sensitive tasks, started to face sophisticated attacks [TSS17] as they became more widely used. As attackers continued to evolve their methods, these threats also began to target REEs [Bos+16], which are the standard operating environments where most applications run. The adaptation of these attacks highlighted the need for comprehensive security strategies that not only protect SEs and TEEs but also ensure the integrity and security of the entire CoT, including the REE.

Moreover, the general public increasingly relies on digital technologies for daily operations such as banking, shopping, and communication. This widespread adoption of digital services amplifies the importance of having a secure CoT. Ensuring that all components of the CoT are robust against security threats is crucial to protect users' sensitive data and maintain trust in digital systems.

## 1.3 Challenges

My research activity focuses on improving the security features introduced in Figure 1.2. These activities are divided into three main challenges.

**Research question 1: How are local and platform security functions, provided by hardware RoT, developed and used to enhance security?** During my Ph.D. thesis [Bou14], I focused on the security of embedded software implementations within SEs. Following this, my research expanded to

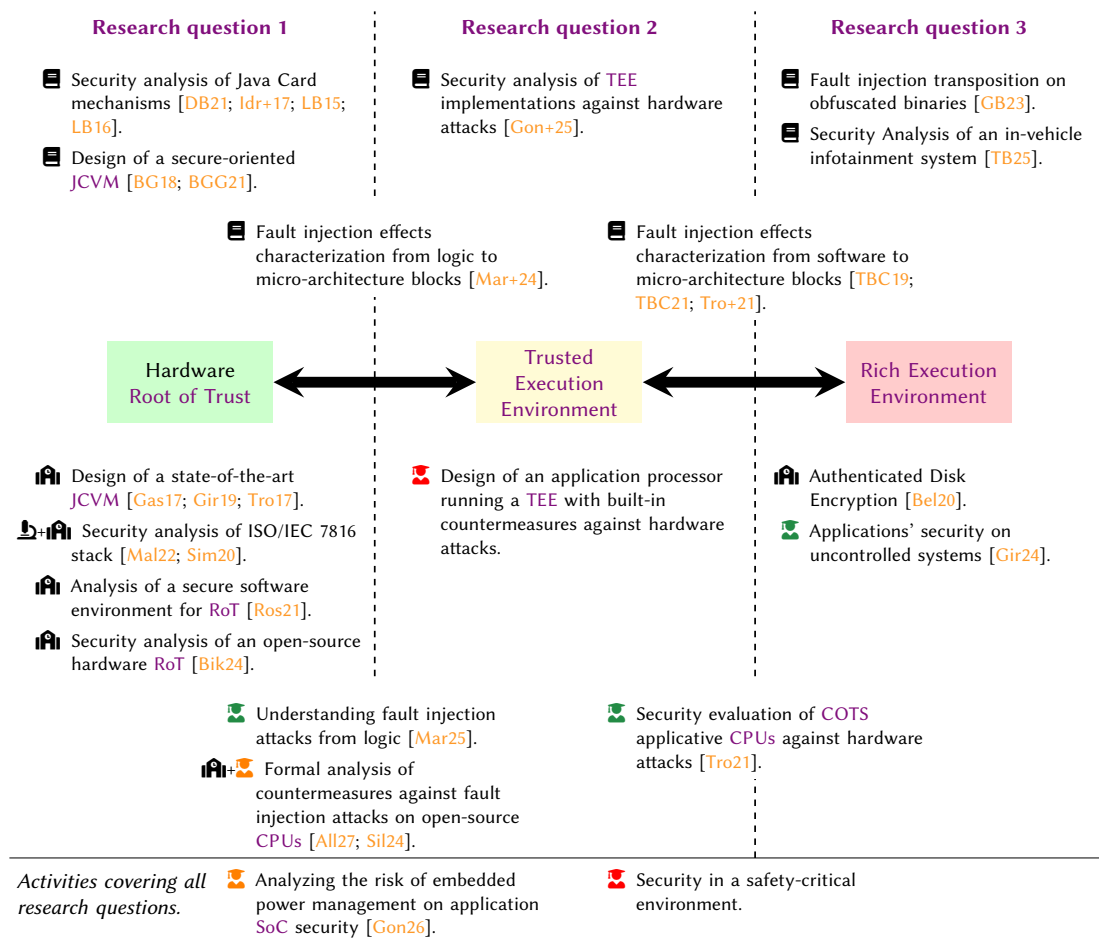
explore the broader use of hardware **RoTs**. This involved studying how hardware **RoTs** are integrated inside a **CoT** from both developer and user perspectives. My goal is to anticipate potential risks and understand the critical role hardware **RoTs** play in maintaining security, ensuring that they are effectively utilized to safeguard sensitive operations.

**Research question 2: How can high security levels be achieved in **TEE**, which provide high-performance environments for *business security functions*?** The advent of **TEEs** in application **CPUs**, such as Arm TrustZone [Arm17], has significantly enhanced security for sensitive applications requiring performance. In 2015, the ANSSI certified the **TEE protection profile** [Glo20b], establishing a framework for evaluating the hardware and software implementations of **TEEs**. This certification marked an important step forward, despite the framework not fully covering hardware attacks. Concurrently, hardware attacks emerged as a notable threat [MBB16; TM17; TSW16; Vas+20] against application **CPUs**. These attacks, adapted from those traditionally targeting hardware **RoTs**, exploit the broader attack surface of application **CPUs**, making their protection more challenging.

**Research question 3: How can sensitive applications with security features run in the **REE**, an environment focused on *business logic without specific security functions*?** Often, third-party developers lack access to **TEEs** or hardware **RoTs** due to the absence of agreements with platform designers. In the **REE**, malicious behaviors such as tampering with the environment or unauthorized access to process memory can occur, making these binaries vulnerable to attacks under a white-box security model [Cho+02]. Consequently, platform users who install applications from the **REE** app store or manually often encounter obfuscated binaries. Such obfuscated binaries are commonly protected by **DRM**-based mechanisms, deployed for instance for video games, which provide anti-piracy and anti-cheating protections [KAS20; Yah23], and for audio and video streaming applications [App16; Wid17]. It appears also that some payment applications, ideally secured by **TEE** or hardware **RoT**, are deployed in the **REE** due to contractual limitations, significantly elevating security risks, a concern acknowledged by the Payment Card Industry [Pay20]. The scientific community actively monitors these applications to detect vulnerabilities first [Bar+22; Heu24; PSF22]. According to Thomas [Tho22], the security race between updates and the exploitation of both known and unknown vulnerabilities is critical. Highlighting the significant risks of employing such environments for sensitive tasks underlines the significance of my prospective research to evaluate and mitigate these risks. This area is interesting for assessing the viability of using obfuscated binaries as a **RoT** in the **REE**.

## 1.4 My Contributions

To provide a global overview of my research activities, Figure 1.3 displays my significant publications along with the supervisions (Ph.D. theses, apprenticeships, and internships) I have participated in.



**Figure 1.3** – Overview of my research activities, based on Figure 1.2. This figure shows my publications (indicated by the 📖 icon) at the top, categorized according to CoT elements, and various supervision activities at the bottom. Regarding supervisions, internships are represented by 🏠, apprenticeships by 🎓, and ongoing Ph.D. supervisions by the orange 👤 icon. Completed Ph.D. theses are marked by a green 🎓 icon. Finally, the red 🚨 icon represents Ph.D. projects that I plan to initiate as soon as possible.

During my research activities, I address simultaneously studies on each CoT element, engaging in *applied research* where I cover each challenge introduced in Section 1.3.

### 1.4.1 Contributions to Hardware Root of Trust Security

My research was initially focused on the security of software layouts embedded in hardware RoT, specifically on SE RoT implementations. Building upon the SE foundations laid during my Ph.D. thesis [Bou14; Idr+17], I examined the security of Java Card platform implementations, which are extensively embedded in SE. I pushed further these studies to scrutinize the security of each mechanism interacting with the Java Card platform, as outlined in the Java Card protection profile [Ora21c]. Particularly, I investigated the security of the compilation process [DB21] and the application verification process [LB15;

LB16] employed prior to loading applications. These studies uncovered vulnerabilities and eventually raised the Java Card platform security. Additionally, I studied the security of the communication protocol implementation, specifically the ISO-7816 stack, during the work with Simunovic’s apprenticeship [Sim20] and Malki’s internship [Mal22]. I also investigated the security of CPU implementations against fault injection attacks with the supervision of Silva Araújo’s internship [Sil24].

During security evaluations, I observed a trend among developers to favor organizational processes over technical solutions to mitigate security risks. Advocating for integrating protections directly into the execution environment, I aimed to achieve security that is less dependent on human factors. In the absence of an open reference implementation of the Java Card specification, I initiated the development of a state-of-the-art and modular Java Card platform version. This project was carried out during the internships of Gaspard [Gas17] and Giraud [Gir19]. Due to licensing constraints the source code is not public. This implementation includes innovations in security [BG18], memory management, and the compilation process [BGG21]. During Troughkine’s internship [Tro17], we implemented a dedicated hardware execution environment to assess the potential performance and security benefits of running Java Card applications natively. This involved developing a processor capable of directly executing Java Card bytecode as assembly, thereby opening new avenues to optimize both the execution efficiency and security of the Java Card platform.

## 1.4.2 Contributions to Trusted Execution Environment Security

Next, my research expanded to studying the security of TEE implementations embedded in application CPUs, a subject initiated during Troughkine’s Ph.D. thesis [Tro21]. This work aims to understand how hardware attacks can extract secrets from the TEE. While the TEE protection profile [Glo20b] acknowledges the need to address hardware attacks, no commercially deployed TEE implementation currently integrates countermeasures against them.

In Troughkine’s Ph.D. thesis [Tro21], we focused on assessing the potential and effects of fault injection attacks on application CPUs [TBC21; Tro+21]. Since we analyzed fault effects on COTS components without access to their internal implementations, we characterized these effects from the CPU’s instruction set down to its micro-architecture blocks [TBC19]. During Marotta’s Ph.D. thesis [Mar25], we further explored the impact of faults from a hardware logic perspective [Mar+24].

To advance this research, I aim to bridge the gap between the work initiated with Troughkine and that conducted Marotta to achieve a comprehensive understanding of fault effects on application CPUs. This knowledge will be instrumental in developing efficient countermeasures spanning both software and hardware layers.

In parallel, Alle Monne’s Ph.D. thesis [All27] focuses on evaluating the effectiveness of embedded countermeasures in hardware RoTs. His work represents a key step in assessing the security of application processors running a TEE and their resilience to fault injection attacks. In this study, we analyze the resistance of these implementations directly from their hardware source code, allowing for a deeper assessment of their robustness and potential vulnerabilities.

Additionally, in Gonidec’s Ph.D. thesis [Gon26], we examine the implications of integrated power management modules on the security of TEEs [Gon+25]. By investigating how these modules can be manipulated to induce faults or leak sensitive information, we aim to identify vulnerabilities and propose mitigation strategies. The findings from this research could be extended to all hardware blocks embedded in application SoCs, including hardware RoTs and REEs, broadening its impact beyond power management security.

### 1.4.3 Contributions to Rich Execution Environment Security

Finally, my research has broadened to analyzing sensitive applications running in the REE. These applications often lack access to hardware RoT or TEE, making them susceptible to white-box attack models [Cho+02]. In this context, during Giraud’s Ph.D. thesis [Gir24], we explored the security of software in these environments. Our studies primarily focused on obfuscated applications, investigating them under the scope of reverse engineering tools where anti-reverse and anti-debug protections are implemented. Together with Giraud, we analyzed the feasibility and implications of transposing hardware attacks, mostly studied against hardware RoT, onto an obfuscated binary [GB23].

## 1.5 Organization of this Manuscript

This manuscript addresses the three challenges introduced in Section 1.3 through the following research questions. It is organized as follows:

- Chapter 2 describes my research activities related to the software layout embedded in hardware RoT. Extending the work begun during my Ph.D. thesis [Bou14], this chapter analyzes each component interacting with the RoT runtime environment, from the compilation process to the application and external communications of the RoT. It addresses the first research topic: *How are local and platform security functions developed and used to enhance security?*
- Chapter 3 discusses my activities about the TEE environment, particularly focusing on application CPU defenses against hardware attacks. Despite the current TEEs not being designed to be resistant against such attacks, the TEE protection profile [Glo20b] suggests they should be. Consequently, this chapter evaluates threats and designs efficient countermeasures that balance performance and security. It goes deeper into TEE security against advanced attackers as outlined by the TEE protection profile, aiming to answer the second challenge: *How can high security levels be achieved in high-performance environments for business security functions?*
- Chapter 4 examines the challenges of executing sensitive applications within the REE, an environment primarily designed for *business logic without specific security functions*. Due to the lack of access to TEEs or hardware RoTs, third-party developers must rely on alternative protection mechanisms, such as software obfuscation, DRM-based solutions, or hardware units provided by the application CPU. In this chapter, we analyze how sensitive applications can operate securely within an untrusted execution environment, evaluating the feasibility and effectiveness of various

protection techniques against potential threats. It addresses the third research topic: *How can sensitive applications with security features run in the REE, an environment focused on business logic without specific security functions?*

- Chapter 5 concludes this manuscript by outlining my future research perspectives.

## Chapter 2

# Contributions to the Hardware Root of Trust Security

In [Chapter 1](#), we explored the essential role of hardware [RoTs](#) as the foundational element in establishing a [CoT](#), crucial for securing computing environments against diverse threats. We now introduce how hardware [RoTs](#) ensure secure operations and protect sensitive data integrity and confidentiality within various systems, thereby enhancing trust in digital platforms.

Building upon these principles, this chapter describes my specific contributions towards advancing the security of embedded software in hardware [RoTs](#). Given their critical function in initiating and maintaining a [TEE](#), it is crucial to ensure the security of hardware [RoTs](#) and to foresee new attack vectors.

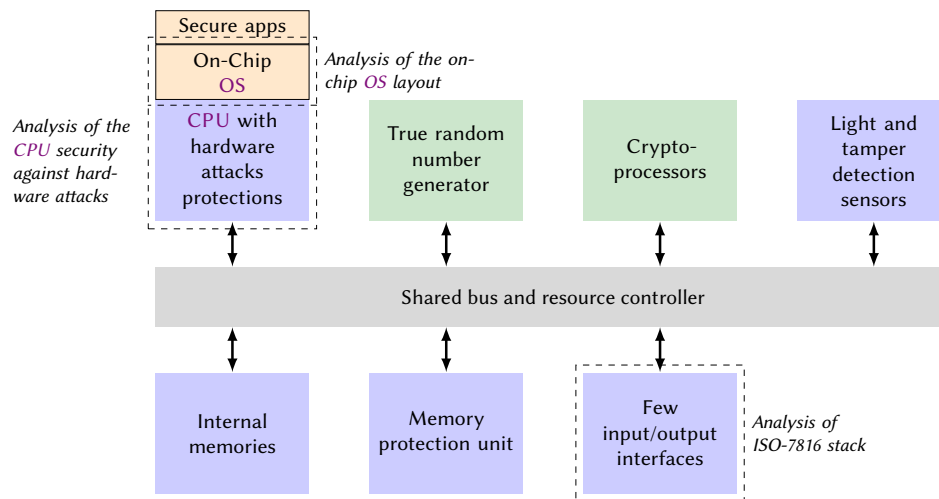
In the field of security technology, [SEs](#) are recognized as fulfilling all the requirements of a hardware [RoT](#). These elements inherently provide a tamper-resistant environment, crucial for secure data storage and cryptographic operations, thereby establishing their keystone in hardware [RoT](#) implementations. As of 2018, GlobalPlatform estimated the deployment of approximately 55 billion [SEs](#) worldwide [[Glo18a](#)]. Moreover, both the hardware and software components of [SEs](#) undergo rigorous security evaluations under the Common Criteria scheme [[Int22b](#)]; each evaluation taking at least six months to one year with several experts involved. By 2023, about 200 products had been evaluated under this scheme within the European Union [[Com24](#)]. Given the vast number of deployed units and the stringent security requirements they satisfy, [SEs](#) is an optimal choice for my research, providing a solid foundation for the widespread application and validation of security principles.

This chapter outlines my contributions to enhancing the *security of software embedded in hardware [RoT](#)* with a main focus on [SE](#) implementations, beginning with an introduction to the hardware and software architectures of [SEs](#) in [Section 2.1](#). This section highlights my advancements in [SE](#) security. The chapter is divided into three main sections: [Section 2.2](#) introduces my contributions to the software layout, specifically focusing on Java Card technology; extensively deployed in [SEs](#). Following that,

Section 2.3 investigates the security of input/output interfaces, analyzing potential vulnerabilities that could be exploited to extract information from SEs. Finally, Section 2.5 concludes this chapter by summarizing the findings and presenting my future directions in the security of hardware RoT.

## 2.1 Common Secure Element Architecture

A SE is designed to resist to both hardware and software attacks, as outlined in its dedicated protection profile [Eur14]. In the event of an attack, it can erase non-volatile memory and enter a permanently disabled state to prevent further access. To achieve this, the SE integrates dedicated security hardware, including a cryptographic coprocessor, a true random number generator, tamper detection sensors, and typically a CPU with built-in protections against hardware attacks [Ars+20; CCH23; Wer+19]. These components are embedded in a low-performance SoC, named a microcontroller, as illustrated in Figure 2.1, and complemented by a hardened software stack. Most SEs also include a JCVm, ensuring a secure runtime environment [ZDS23]. The entire hardware and software stack undergoes rigorous security evaluations to comply with the requirements defined in the SE protection profile [Eur14].



**Figure 2.1** – Overview of the SE hardware and software architecture, adapted from the SE protection profile [Eur14]. Yellow boxes highlight the software layout, blue boxes represent general-purpose hardware, and green boxes indicate cryptographic hardware blocks, which are mainly focus of hardware attack literature [BB23; CX23; Pic+23]. The *Internal Memories* block, shown as a meta-block, may include SRAM, Flash, and OTP memory. Dashed boxes and italicized labels highlight my contributions to SE security.

During a security evaluation, an ITSEF conducts assessments under the supervision of a certification body and according to a public certification scheme, such as Common Criteria, or a private one, like EMVCo. The certification process evaluates the resilience of security functions deployed to protect sensitive assets against a defined level of attacker [Eur14; Eur22]. The target of evaluation and the attacker level are determined by the customer for whom the product under evaluation is developed. In

cases where several developers are involved in the design of a **SE**, the security evaluation can be carried out through composition. This approach involves analyzing one part of the target through an initial evaluation, while other parts undergo subsequent evaluations conducted by one or more **ITSEFs**. The use of composition not only eases a modular approach to security assessments but also permits for the specialization of security evaluations across different system components such as hardware, **OSes**, and applications. This modular method helps breaking down the security problem into small parts, making it manageable and focused on specific functions or layers within the system.

**SEs** which successfully pass security evaluations under the Common Criteria scheme, supervised by a national certification body, can achieve certifications up to the “*Formally Verified Design and Tested*” level [Car07]. In France, the national certification body is **ANSSI**.

Cryptography forms the cornerstone of security within **SE**, playing a critical role in ensuring the integrity, confidentiality, and authenticity of sensitive data. Security evaluations of **SEs** are conducted with respect to the current state of knowledge, which includes both publicly available scientific publications and restricted information. In the security analysis of **SEs**, robustness is primarily assessed in terms of cryptographic security against flawed practices [Nem+17] and hardware attacks [BB23; CX23; Pic+23].

However, the security of the software that utilizes these cryptographic implementations has received comparatively less attention to protect them against hardware attacks. This results in a more limited state of knowledge in this domain compared to the well-studied field of cryptographic implementations. This gap is significant, as these software components are responsible for initializing and managing cryptographic operations. In my research, I have aimed to *address this gap by analyzing, understanding, and improving the software embedded in the SE*.

My research on **SE** security is divided into three main areas: the *security of the on-chip OS*, as introduced in **Section 2.2**, the *security of input/output interfaces*, described in **Section 2.3**, and the *analysis of CPU security against hardware attacks*, presented in **Section 2.4**. In each section, I present my contributions along with associated perspectives.

## 2.2 Security of the On-Chip Operating System Layout

### Associated contextual elements of this research field

- Collaborations:** Jean Dubreuil (SERMA, Pessac, France), Arnaud De Grandmaison (Arm, Paris, France), Said El Hajji (LabMIA, Faculté des Sciences, Rabat, Morocco), Karine Heydemann (LIP6, Sorbonne University, Paris, France), Noredine El Janati (LabMIA, Faculté des Sciences, Rabat, Morocco), Julien Lancia (Thales ITSEF, Toulouse, France), and Jean-Louis Lanet (INRIA-LHS, Rennes, France).
- Publications:** [AG21; BG18; BGG21; DB21; ldr+17; LB15; LB16]
- Supervisions:** Internships of Léo Gaspard [Gas17], Thomas Troughkine [Tro17], Vincent Giraud [Gir19], and Ever Atilano Rosales [Ros21].

In SE security, having a comprehensive view of hardware and software security threats for complex applications is a challenging task. For example, most SEs embedded in smartcards have an EMVCo-compliant [EMV07; EMV08; EMV11a; EMV11b; EMV11c; EMV11d] application. Designing and securing state-of-the-art EMVCo-compliant applications for SE requires considering both hardware and software attacks. Developers of such applications must have multidisciplinary skills and expertise in both hardware and software security. Such a profile is extremely rare and software developers often lack the expertise to design complex applications.

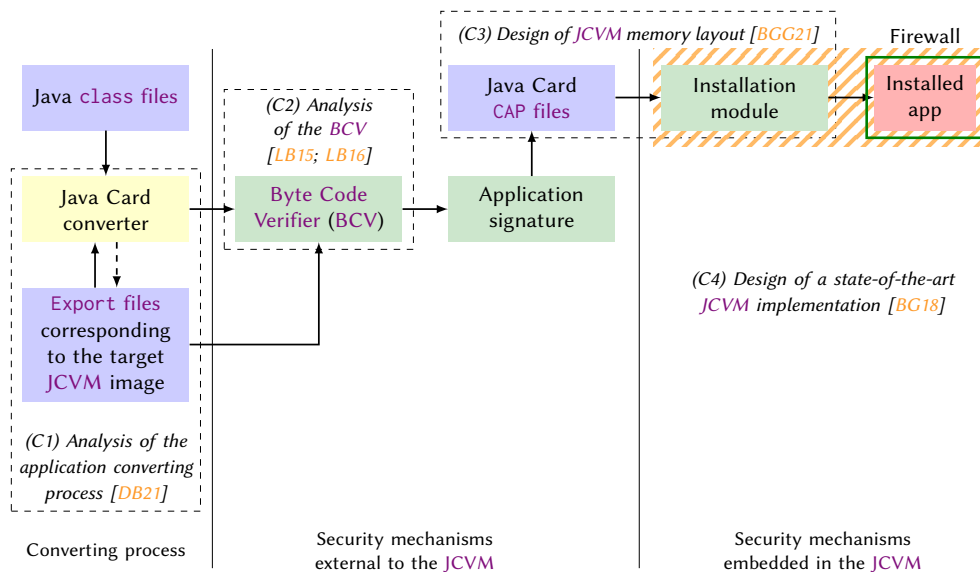
The Java Card technology addresses this need by offering a standardized and secure on-chip OS that abstracts hardware security complexities, thereby enabling developers to focus on their application's features and to define protection requirements more clearly.

Java Card supports the development of secure applications regardless of specific hardware or OS models, enabling application developers to work without knowledge of the underlying platform specifics. This abstraction means that developers can focus solely on functionality and security without adapting to each target's characteristics. As a memory-safe language, Java Virtual Machine (JVM) also provides protections against common memory-related vulnerabilities, such as buffer overflows and invalid memory access, which enhances application security. For its time, Java Card brought the advantages of the Java language (security, portability, and simplified development) into a resource-constrained environment. This approach simplifies the development process, ensures uniform deployment across various devices, and effectively mitigates security risks associated with diverse hardware environments.

The Java Card technology is employed in the vast majority of evaluated SEs, with nearly 100 Java Card products evaluated annually under the French Common Criteria scheme. Since 2015, the Java Card technology has increasingly been targeted towards the automotive and Internet of Things (IoT) security sectors [Pas22]. According to [Pas22], 6 billion devices equipped with a JCVM are deployed annually. Java Card specifications, released and licensed by Oracle, have evolved to the current version 3.2. Throughout my research, I have studied Java Card versions from 2.2.2 to 3.0.5.

### 2.2.1 Overview of the Java Card Technology

For software developers, Java Card follows the same development and compilation process as Java Standard Edition. Developers write applications in Java and compile them into `class` files using the `javac` compiler. As Java `class` files are not suitable for resource-constrained devices like `SE`, they must be converted into smaller `CAP` files using a Java Card converter. This process is shown on the left side of Figure 2.2.



**Figure 2.2** – This figure describes the Java Card architecture, divided into three sections: the conversion of Java `class` files for resource-constrained devices like `SE`, external and embedded security mechanisms within the `JVM`. The color-coded blocks signify: blue for files involved in the compilation process to load applications into the `JVM`, yellow for elements without security features, green for elements with security features, and red for the installed application in a `JVM`. Since this application must access only authorized resources [Ora21c], the `JVM Firewall`, shown as a green rectangle, enforces application segregation within a specific security context.

The security mechanisms are implemented both externally and internally to the `JVM`. Externally, the `BCV` ensures that the structure and semantics of the Java Card `CAP` files comply with Java security rules, such as prohibiting pointer arithmetic. Internally, an installation module loads the Java Card `CAP` file into non-volatile memory, performing few checks close to those conducted externally, along with possibly additional verifications. Once these checks are complete, the installation module converts the `CAP` file into undocumented internal structures used by the `JVM` implementation. Additionally, at runtime, the `Firewall` dynamically verifies access to resources, ensuring application isolation and adherence to security policies [Ora21b].

The common security evaluation target [Ora21c] for `JVM` implementations is marked in an orange hashed rectangle (▨). This part was studied during my Ph.D. thesis [Bou14].

Contributions from my post-Ph.D. work are highlighted in italic, with studied `JVM` blocks shown in dashed boxes and started with the label (C1), (C2), (C3), and (C4).

The embedded `JVM` implementation security relies on the a specific software validation done by the `Byte Code Verifier (BCV)` which checks for type correctness and memory access properties. Few

checks occur at load time before **CAP files** are installed on the device. Most Java Card platforms do not include a **BCV** due to its high memory consumption, so verification is performed outside the device by the issuer or a trusted third party, as noted in the middle of [Figure 2.2](#).

In addition to static verification, the Java Card Firewall ensures application isolation at runtime. It creates separate security contexts for each package, preventing unauthorized access between packages. This is depicted on the right side of [Figure 2.2](#).

The Java Card architecture, along with the contributions I made after my Ph.D. thesis, are illustrated in [Figure 2.2](#). During my doctoral research, we analyzed embedded implementations [[Bou14](#); [ldr+17](#)]. Subsequently, I extended this work to focus on external security mechanisms for **JCVM**, as detailed in [Section 2.2.2](#).

## 2.2.2 Analysis of the Java Card External Mechanisms

This section explores external mechanisms that protect **JCVM** implementations from software attacks and analyzes critical elements of Java Card security, which are often assumed to be trustworthy.

During my [Ph.D. thesis](#) [[Bou14](#)], we highlighted the **BCV** as a critical element of **JCVM** security. The **BCV** performs crucial security checks to ensure each embedded application complies with Java security rules. It targets the **CAP file** with two main checks: *type correctness*, which prevents prohibited type conversions through abstract interpretation, and *structure verification*, ensuring compliance with Java Card specifications [[Ora21b](#)]. Even a minor unchecked element in the **CAP file** can introduce significant security vulnerabilities into **SE**, as demonstrated in [[FV10](#)].

### 2.2.2.1 Security Analysis of the Byte Code Verifier

#### Related publications

- 📄 Julien Lancia and Guillaume Bouffard. “Java Card Virtual Machine Compromising from a Bytecode Verified Applet”, In: *CARDIS 2015* [Article in PDF](#) [[LB15](#)]
- 📄 Julien Lancia and Guillaume Bouffard. “Fuzzing and Overflows in Java Card Smart Cards”, In: *SSTIC 2016* [Article in PDF](#) [[LB16](#)]

Testing software thoroughly is complex, as it requires ensuring both test coverage and compliance of the code with the implemented specification. Efforts have been made to characterize the **BCV** of Java Standard Edition from both functional and security perspectives. The **JVM** dynamically calls the Java **BCV** during the loading of class files, verifying compliance and ensuring security before execution. For example, Siret [[Sir99](#)] used code mutation and a reference **JVM** with **BCV** as an oracle for automatic test case generation. Similarly, Calvagna and Tramontana [[CT13](#)] developed a formal **JVM** model that includes **BCV** and used model-based testing to evaluate compliance.

The Java **BCV** and the Java Card **BCV** serve different purposes due to the distinct file formats and security constraints of Java Card compared to Java. Java Card supports a simplified subset of Java, but the Java Card **BCV** statically verifies that the application complies with security rules and adapts to the

targeted **JVM** implementation security constraints. In contrast, the Java **BCV** is more generic, as it dynamically verifies each loaded **class file**, it can rely on the execution context to ensure compliance with security rules.

In the Java Card technology, implementations of both external [Ler02] and embedded [Ber+14; Cas02] Java Card **BCV** have been developed from formal specifications. These models are also used to test some part of the Oracle's closed-source Java Card **BCV** [CFT14; SFL13] implementation provides within the Java Card development kit; however, no comprehensive work has fully targeted the Java Card **BCV**.

Faugeron and Valette [FV10] analyzed Oracle's Java Card **BCV** version 2.2.2 using non-formal methods. Their work revealed a flaw in the interpretation of branch instructions, leading to undetected type confusion. This issue was subsequently patched in version 3.0.3. Their findings raised questions about the potential existence of other incorrect verifications not detected by the formal models used in the analysis, highlighting the need for further investigation into such vulnerabilities.

**A Missing Check in the Byte Code Verifier** To assess the application verification process by the **BCV**, we developed a fuzzing approach based on genetic mutation. Starting with a simple, legitimate application, we introduced mutations by randomly altering a byte in the application's **CAP file** for each test iteration. The **BCV** then evaluates whether the mutated **CAP file** remains valid. If it passes, the **cref**<sup>1</sup> attempts to execute the application associated to the mutated **CAP file**. Errors thrown by **cref** alert on the presence of a malformed application, which may expose security vulnerabilities. These **CAP files** undergo further detailed analysis. Employing this method, we uncovered a significant vulnerability across versions 2.2.2 to 3.0.5 of the **BCV** [LB15]. This contribution is labeled by (C2) in Figure 2.2.

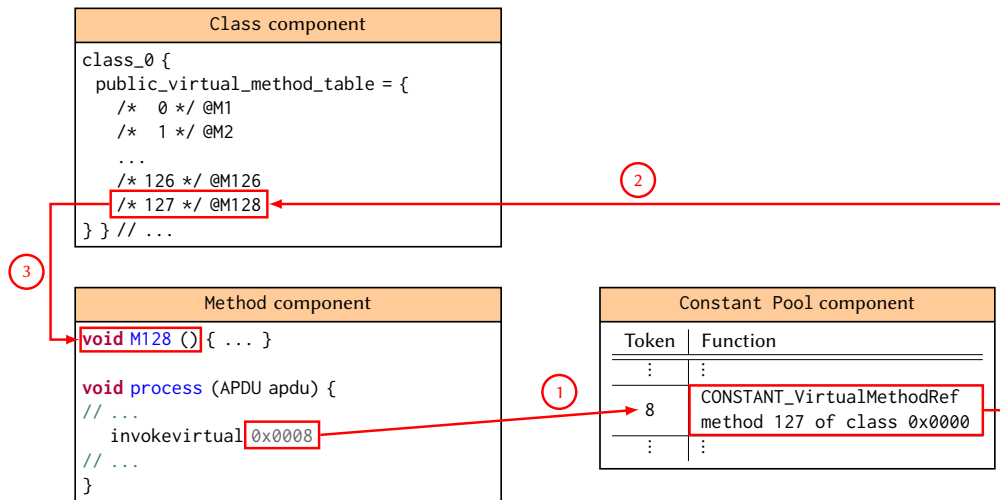
During the method resolution process, the runtime environment translates a token into an address, as detailed in Figure 2.3. Our testing approach revealed a critical missing check in the **BCV**: if an entry in the public virtual method table is missing, the **BCV** fails to raise an error. Some **JVM** implementations may attempt to resolve this missing address by overflowing the public virtual method table, using the subsequent two bytes for address resolution. As demonstrated in [LB15], an attacker can exploit this behavior to redirect application control flow to a malicious payload.

This work was extended in [LB16], where techniques for executing polymorphic code were developed to conceal a payload within an application. This strategy becomes essential for an attacker when a third party performs comprehensive verifications on the application beyond the standard **BCV** checks, including specific requirements dictated by the target platform's security guidelines. Such an attack underlines the importance for third parties to conduct expensive verifications before loading applications onto a Java Card-based **SE**.

**Security Impact of a Missing Check in the Byte Code Verifier** The identified vulnerability involved a missing field check in the **CAP file** by the **BCV**, potentially exploitable on various **JVM** implementations. We provide an attack path on a deployed **JVM** implementation [LB15]. Due to many **SE** devices does

---

1. The **cref** is a reference Java Card simulator provided in the **Java Card Development Kit (JCDK)** and serves, in this work, as our fuzzing oracle.



**Figure 2.3** – This figure illustrates the Java Card method resolution process within a CAP file that encompasses Class, Method, and Constant Pool components, each essential to the operation. The Method component lists bytecode tokens; the Constant Pool provides linkage details for each token, specifying package, class, and method information. Resolution occurs when the JCVM retrieves the Constant Pool entry for a token, such as 0x0008, directing to the appropriate method in the public virtual method table field of the Class component. Red arrows marked as ①, ②, and ③ indicate the order of resolution.

not embedding a comprehensive and evaluated BCV, the embedded JCVM relies on external tools to ensure that applications meet Java Card specifications without doing on itself these checks. Oracle’s BCV, adhering closely to Java Card standards, is widely used for this purpose.

We initiated a responsible disclosure with Oracle in early 2015. Oracle responded by releasing JCDK version 3.0.5u1 on August 19, 2015, and informed their customers. Conforming to the Java Card protection profile [Ora21c], all SE must use the latest BCV version. In compliance with Common Criteria procedures, certification bodies conducted pre-disclosure verifications on evaluated platforms. After confirming the mitigation of the vulnerability, we publicly disclosed the details.

### 2.2.2.2 Security Analysis of the Class to CAP file Converting Process

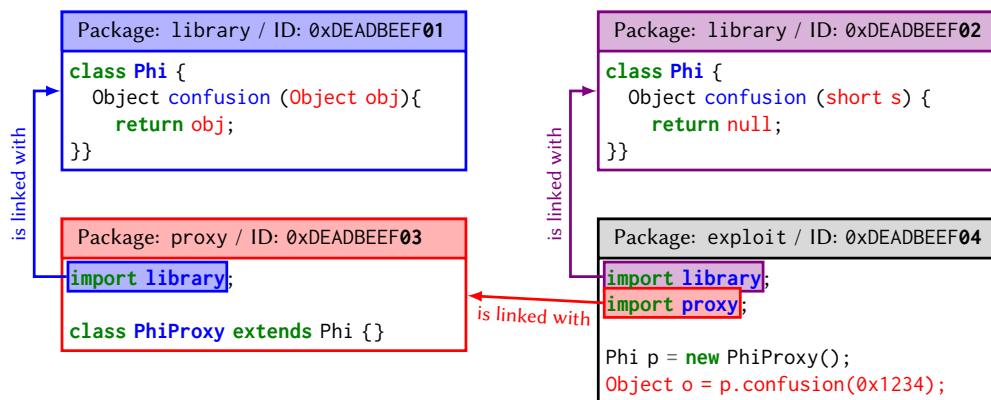
#### Related publication

📖 Jean Dubreuil and Guillaume Bouffard. “PhiAttack - Rewriting the Java Card Class Hierarchy”, In: *CARDIS 2021* [Article in PDF](#) [DB21]

To perform the analysis of a CAP file, the BCV requires all export files from the target JCVM that will be imported by the application under review. However, export files lack integrity and authenticity assessment, posing security risks when validating potentially corrupted information. We highlighted these risks in [Bou+13a], noting consequences to counterfeit export files. We subsequently demonstrated a Man-in-the-Middle attack that exploits malicious export files to extract cryptographic keys. This involved installing a backdoored Application Programming Interface (API) on a JCVM, enabling it

to intercept and record keys generated by legitimate cryptographic APIs. However, this attack is easily detectable and thus not considered realistic.

**Ambiguity in Method Resolution Process** In [DB21], we expanded our analysis to explore how the Java Card converter translates symbols from class files to CAP file. Symbols in class file are encoded as Unicode strings, while in CAP file, they are represented by 1- or 2-byte token values. We identified an ambiguity in the method resolution process that can be exploited in several JVM implementations. This vulnerability, undetectable by the Java Card BCV, can be leveraged within legitimate uses of the JCDK. Refer to Figure 2.4. This contribution is named (C1) in Figure 2.2.



**Figure 2.4** – This figure illustrates a potential ambiguity in method resolution within Java Card platforms. The exploit package imports both the library package (purple, top right) and the proxy package (red, bottom left). The proxy package, in turn, is linked with a different library package (blue, top left), highlighting possible confusion during linking. At runtime, due to an implicit cast from the PhiProxy class to the Phi class, it becomes unclear whether a confusion method call in the exploit package is intended for the purple library or the blue library. This illustrates a security challenge in ensuring the correct method linkage.

In Java Card platforms, every package is identified by a unique identifier. Currently, there are no measures in place to prevent a developer from creating a package that shares a name with an existing one, as long as the new package’s identifier is not already in use. Both during compilation and at runtime, this situation is manageable: the BCV can identify and differentiate between the two packages to ensure their correct usage. The JVM interprets bytecode based on the content of CAP files, which import packages using their identifiers.

The code architecture shown in Figure 2.4 is accepted by the BCV. Both the blue and purple library packages include a confusion method with different signatures; one accepts an Object as parameter, the other a short. The BCV associates the confusion method in the exploit package with the purple library based on its export files, unaware that the proxy package links to a different library version. This misassociation causes the BCV to verify the method’s signature incorrectly. When the confusion method is invoked, the JVM dynamically resolves the call to the blue library via the class reference in the proxy package, leading to an erroneous execution of the blue library’s method. As detailed in

Figure 2.4, this flaw enable us to craft a pointer from address 0x1234, exploiting the vulnerability.

**Security Impact of Ambiguity in Method Resolution Process** This section explains how the missing informations in both **CAP files** and **export files** can be exploited to circumvent **BCV** checks during method resolution. From the perspective of the Java Card converter, **export files** lack sufficient information to explicitly specify which packages are imported in **CAP files**, preventing the **BCV** from verifying correct method usage according to their signatures. The **BCV** delegates this verification to **JCVM** implementations, which in turn assumes that the **BCV** has already performed these checks.

Such vulnerabilities could allow an attacker to execute malicious code inside a verified application, undermining the Java Card security model. However, exploiting this vulnerability requires additional conditions, such as executing specific malicious payloads on targeted devices, as the only presence of overflow or underflow does not automatically imply the existence of an exploitable attack vector

One way to mitigate this issue is to restrict the use of **export file** format version 2.3, introduced in Java Card 3.1 revision, released a few months before our publication [DB21]. This updated **export file** format includes additional information about, for each class, the imported packages and version, addressing the issue identified. However, the latest versions of the **BCV** still accept **export files** from version 2.2. Enforcing the **BCV** to only accept **export files** from version 2.3 is not currently feasible. Notably, sensitive **export files**, such as those from GlobalPlatform [Glo12], are only available in version 2.2, posing a significant challenge for compliance and security.

The discovery of this vulnerability in **export files** undermines Java Card security and opens up new research directions for identify other potentially missing or incomplete checks. In line with our responsible disclosure policy, all Java Card platform developers potentially impacted by this vulnerability were informed through the Common Criteria scheme prior to public disclosure.

### 2.2.3 Synthesis and Perspectives

This section has presented my contributions to the security of on-chip **OS** embedded in **SE**, with a primary focus on the Java Card platform's security. During security evaluations, a common trend I observed was the preference for organizational processes over technical solutions to mitigate security risks. To counter this, I advocate for integrating security measures directly into the execution environment to reduce dependency on human factors.

#### 2.2.3.1 Perspectives in Java Card Platform Security

##### Related publications

📄 Guillaume Bouffard and Léo Gaspard. “Hardening a Java Card Virtual Machine Implementation with the MPU”, In: *SSTIC* 2018 [Article in PDF](#) [BG18]

📄 Guillaume Bouffard, Vincent Giraud, and Léo Gaspard. “Java Card Virtual Machine Memory Organization: a Design Proposal”, In: *arXiv* 2021 [Article in PDF](#) [BGG21]

Given the absence of an open reference implementation for the Java Card specification, I started the development of a state-of-the-art and modular **JCVM**. This project began during the internships of **Gaspard** [Gas17] and **Giraud** [Gir19]. Two significant results have been achieved.

First, advancements in the security of **OS** running the **JCVM** are presented in [BG18], highlighted as (C4) in Figure 2.2. This work explores how a **Memory Protection Unit (MPU)** can improve the Java Card Firewall to ensure applet segregation. We leveraged the **MPU** to segregate each applet within its own security context, preventing unauthorized access between applets. Most existing Java Card Firewall implementations do not rely on hardware mechanisms to enforce segregation. Developing a hardware-based Java Card Firewall implementation significantly enhances both security and performance.

Second, we proposed a **JCVM** memory layout [BGG21], labeled as (C3) in Figure 2.2. The **JCVM** memory layout is excluded from the Java Card specification and is typically implemented differently by each developer. To design a secure-oriented **JCVM**, it is necessary to have a reference implementation for each part of the **JCVM** to carefully evaluate and improve its robustness. Unfortunately, due to licensing restrictions associated to the Java Card specifications [Ora21a; Ora21b], the source code cannot be shared.

Future development of our **JCVM** implementation will need to enhance tamper resistance against hardware attacks, providing application developers, often not experts in hardware security, the necessary interfaces to protect their applications. Since Java Card 3.0.5, the introduction of the **API SensitiveArrays** and **SensitiveResults** classes have aimed to associate security properties with Java Card class variables. However, these classes are primarily implemented at the software level by the **JCVM** and lack reliance on underlying hardware security mechanisms. They currently support only specific Java Card types, which could be expanded through annotations, allowing developers to specify more precisely the expected security properties of their programs. These **APIs** currently offer only integrity protection, the inclusion of confidentiality is also crucial and should be considered in future revisions.

During **Trouchkine**'s internship [Tro17], we explored the feasibility of running Java Card applications directly in a dedicated hardware environment. We modified an open-source Java processor [Sch05], capable of natively running Java bytecode, to execute Java Card bytecode while respecting its specific constraints. This approach improved performance and security by incorporating hardware security features directly into the Java Card runtime. This design enables the integration of hardware-based security with a strong execution platform, greatly enhancing overall security. Notably, commercial processors running Java bytecode as assembly language exist, such as Sun's Java Processor [Tur96], **JStik** by ajile Systems [Bin+08], and Arm microcontrollers with the **Jazelle** extension [Arm12]. Direct native execution of Java Card bytecode is not documented in the literature, highlighting the need for further research and development to maximize its potential.

### 2.2.3.2 Perspectives in Hardware RoT Security

While *SEs* represent robust implementations of hardware *RoT*, there exist less secure implementations evaluated only for their hardware layer [Alb21]. A *SE* is a hardware *RoT* where both the hardware and software components have undergone a security evaluation. In contrast, a *secure-oriented microcontroller* is a hardware *RoT* where only the hardware has been evaluated. Secure-oriented microcontroller is commonly used in secure boot processes (see Figure 1.1a and Figure 1.1b) as the Google’s Titan [BV21].

Emerging initiatives like the *TrustedFirmware* project [Leg23] aim to provide open-source software-oriented security to establish a *CoT* for *COTS components*. This project includes a bootloader based on *MCUBoot*, which offers firmware updates with signature verification and robust protections against hardware attacks for secure-oriented microcontrollers.

During *Rosales’s* internship [Ros21], we conducted an analysis of *MCUBoot* under potential fault injection attacks, specifically focusing on how compilation parameters can influence the effectiveness of security countermeasures. The results of this work, presented by Arm [AG21] underlined the critical role of the compilation process in fortifying software against hardware attacks. It also emphasized the ongoing need for research to develop a secure *RoT* that substantially enhances the overall security framework of the *CoT*.

## 2.3 Security Analysis of the Input/Output Interfaces

### Associated contextual elements of this research field

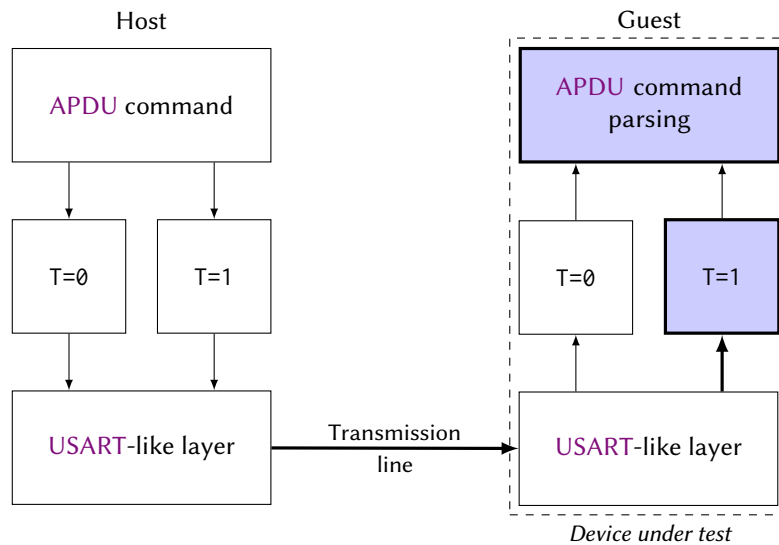
**Supervisions:** Boris Simunovic’s apprenticeship [Sim20] and Louisa Malki’s internship [Mal22].

The study of the security of embedded software implementations extends beyond the software layer; it is also important to analyze the protocols at the input/output interfaces. These protocols, which are exposed by a hardware *RoT*, can be exploited by a malicious user.

A *SE* generally embeds the ISO/IEC 7816 communication protocol [Che00]. ISO/IEC 7816 [Int15] is a client-server type protocol used to communicate with smartcards, and more recently, with contactless devices [Wik24b]. In this protocol, the host functions as the client, sending requests, and the guest acts as the server, responding to these requests. Within this protocol, applications communicate via an *Application Protocol Data Unit (APDU)* command [Int13].

To be transmitted, an *APDU* command is encapsulated over transmission layers named  $T=0$ ,  $T=1$ , or  $T=CL$ .  $T=0$  and  $T=1$  are layers of the ISO/IEC 7816 protocol stack [Int06] and are transmitted over a *USART*-like physical connection. An ISO/IEC 7816 host-guest architecture with  $T=0$  and  $T=1$  layers is depicted in Figure 2.5.  $T=CL$  corresponds to a full protocol stack associated with ISO/IEC 14443 [Int18] for transmission using an NFC modem.

$T=0$  is a character-level transmission protocol where each byte of the *APDU* command is sent as-is.  $T=1$  is a block-level transmission protocol, where the *APDU* command is encapsulated within a more complex block that includes an integrity checksum. Unlike  $T=0$ , the  $T=1$  layer uses several complex



**Figure 2.5** – ISO/IEC 7816 protocol layers. Arrows illustrate the transmission of an APDU from the host sending the command to the guest parsing the received command. The APDU can be transmitted via either the T=0 or T=1 layers. The blue boxes are analyzed under our work.

packet types to manage communication. Due to the complexity of the protocol, understanding and implementing it can lead to potential errors.

### 2.3.1 Security analysis of Embedded ISO/IEC 7816 implementations

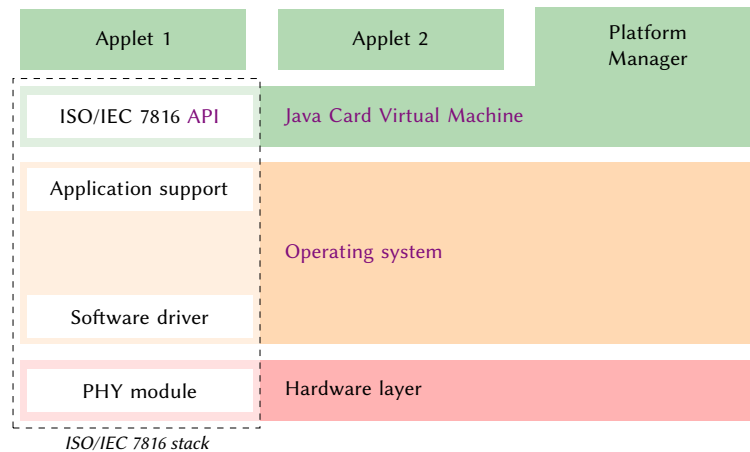
In 2015, Vinet [Vin15] presented an attack on an implementation of the ISO/IEC 7816 protocol within a SE, even though this protocol stack is typically verified during security evaluations. Following the publication by Vinet, it raised concerns about potential vulnerabilities in other deployed and certified implementations. In the state of the art, the focus is on the proper implementation of application protocols [AF18; Lan+18], such as EMV [AVR14; Lan11a] and HTTP [Bar+11]. In 2016, a French ITSEF demonstrated the possibility of exploiting JCVN implementations through flaws in an improperly implemented ISO/IEC 7816 stack; however, the details of this work have not been made public.

In this context, Simunovic began his apprenticeship [Sim20], focusing on studying the compliance of ISO/IEC 7816 stack implementations embedded in SE. This work involves examining various aspects of the ISO/IEC 7816 protocol implementation, as presented in Figure 2.6.

During Simunovic's apprenticeship, we evaluated the security of T=1 implementations embedded in a SE, as indicated by the areas in blue in Figure 2.5. Given the greater complexity of this protocol compared to T=0, we hypothesized that errors might be present in its implementations.

As a first step towards understanding how this protocol works, we implemented the host version of the ISO/IEC 7816 protocol. This implementation is open-source and available on ANSSI-FR/Open-ISO7816-Stack's GitHub repository.

A second step, to evaluate the security of ISO/IEC 7816 T=1 implementations embedded in SE, we



**Figure 2.6** – Overview of ISO/IEC 7816 stack architecture, embedded in a SE, upon each software level. Generally, this kind of implementation is divided into four levels: 1) a *PHY module* implementing physical layer of ISO/IEC 7816 stack, 2) a *software driver* allowing the OS to use the PHY hardware module, 3) an *application support* exposed by the OS to the userland applications and implementing the ISO/IEC 7816 protocol, 4) and a *Java Card API* enabling the applications to use the ISO/IEC 7816 support.

employed a *fuzzing* approach based on model mutation.

Operating with a black box approach, we were unable to instrument specific parts of the protocol within the device under test. Therefore, to test the T=1 protocol as shown in Figure 2.5, we used all parts of the protocol, from the physical layer to the APDU layer. Additionally, our method required an application that uses APDU to ease exchanges with the fuzzer. This necessitated having the target in development mode, where we could load our application. Ultimately, our goal was to test the entire ISO/IEC 7816 protocol stack on SE without the ability to load applications.

During Simunovic’s apprenticeship [Sim20], we modeled the T=1 protocol of ISO/IEC 7816 [Int06] and adapted the *fuzzer* Boofuzz<sup>2</sup>. To evaluate the guest system’s behavior under test, we used a bridge based on an STM32 board, which converts the data generated by Boofuzz into valid or invalid ISO/IEC 7816 commands. This bridge embeds a state machine specifically designed to allow Boofuzz to comprehensively test each part of the ISO/IEC 7816 protocol, providing direct access for Boofuzz to each protocol step. We made necessary adaptations to Boofuzz to integrate it seamlessly with our state machine architecture. The code for the bridge is available on ANSSI-FR/cardstalker’s GitHub. The setup’s architecture is depicted in Figure 2.7.

We used our fuzzing architecture on several smartcards from various manufacturers. The tested smartcards were purchased from public stores. We did not discover any vulnerabilities. However, we did identify behaviors that allowed us to distinguish different implementations through fingerprinting. The ability to use fingerprinting to potentially identify the developer of a specific implementation provides us with ideas for future research directions. These ideas are described in Section 2.3.2.

2. Boofuzz is an open and free *fuzzer* available on [jtpereyda/boofuzz’s GitHub](https://github.com/jtpereyda/boofuzz).

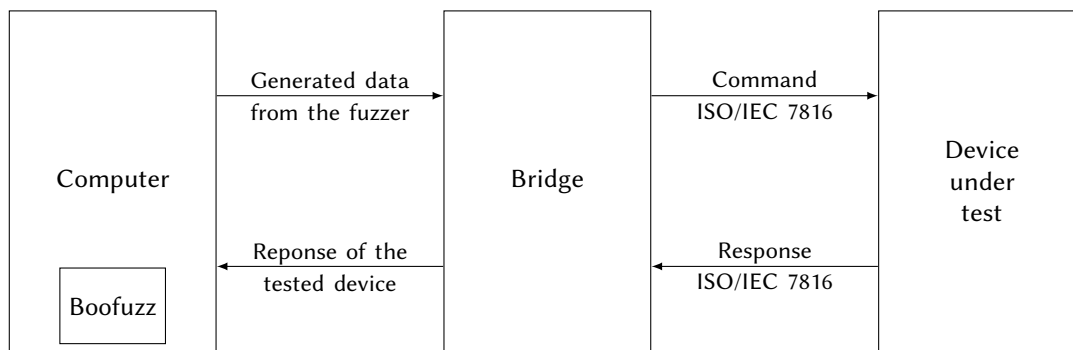


Figure 2.7 – Fuzzing architecture of ISO/IEC 7816 protocol.

### 2.3.2 Synthesis and Perspectives

During [Simunovic's](#) apprenticeship [[Sim20](#)], we developed a fuzzer to assess the robustness of ISO/IEC 7816 implementations at the T=0 and T=1 layers. Our focus during [Simunovic's](#) apprenticeship [[Sim20](#)] was particularly on the T=1 layer. Nevertheless, we did not discover any vulnerabilities in the targeted implementations.

While analyzing the fuzzing results, we discovered that targeted [SE](#) devices, whether from different manufacturers or different models from the same manufacturer, responded differently to specific ISO/IEC 7816 T=1 commands. This variability in responses can be used to fingerprint devices, potentially revealing information about the developers of the [JVM](#) platform and the installed applets. Furthermore, [SE](#) devices with identified vulnerabilities [[Nem+17](#); [Roc+21](#)] can be fingerprinted and potentially exploited in environments where the [SE](#) is anonymized.

Traditionally, fingerprinting has been a concern primarily for web browsers [[Lap+20](#)] and [OS](#) [[SCW19](#)]. This approach has recently been extended to fingerprint cryptographic implementations embedded in [SE](#), as demonstrated by a study focused on Java Card applets [[Sve+22](#)]. Such research typically requires the ability to load applications onto the [SE](#) to identify specific implementations. The capacity to load applications often provides comprehensive information about the platform. Moreover, the capability to fingerprint the ISO/IEC 7816 protocol enhances the potential to gather detailed information from a target [SE](#) even without specific applications installed.

This topic was central to [Malki's](#) internship [[Mal22](#)], where we analyzed the differences in ISO/IEC 7816 responses to categorize target implementations by developer and version. This work is ongoing, and further analysis is needed to systematically organize and interpret the fingerprinting results obtained.

## 2.4 Analysis of CPU Security Against Hardware Attacks

### Associated contextual elements of this research field

- Collaborations:** Luc Bonnafox (ANSSI), Damien Couroussé (CEA/List, Grenoble, France), Mathieu Jan (CEA/List, Saclay, France), and Simon Tollec (CEA/List, Saclay, France).
- Supervisions:** The internships of [Mário da Silva Araújo \[Sil24\]](#) and [Angie-Sofia Bikou's](#) internship [[Bik24](#)] and [Jonah Alle Monne's](#) Ph.D. thesis [[All27](#)].

[CPUs](#) in [SE](#) are designed to be resilient to both hardware and software attacks, including fault injection attacks, which pose a significant threat to the integrity and confidentiality of sensitive data in embedded systems [[Bar+06](#)]. These [CPUs](#) typically include built-in protections against hardware attacks. However, in my work, I have primarily focused on fault injection attacks, while side-channel attacks will be a topic of future research. With the advent of RISC-V [Instruction Set Architecture \(ISA\)](#), it has become possible to conduct a detailed study of the implementation of embedded countermeasures.

Fault injection allows an attacker to force the processor out of its expected functioning bounds, potentially inducing logical changes at both hardware and software levels. These changes can lead to unexpected states or execution paths, which can be exploited in attacks to leak secrets or escalate privileges. While traditional fault injection robustness analyses consider abstract models of the processor's [ISA](#) and fault effects, recent research emphasizes the need to account for fault injection consequences at the micro-architectural level [[Lau+19](#); [Lau+21](#)]. At the hardware level, precise knowledge of the processor's micro-architecture is required, as fault effects often stem from hidden sequential elements like pipeline registers. At the software level, the impact of a fault depends on the execution context, including the nature of program instructions and the state when the fault occurs. Consequently, a comprehensive robustness analysis requires considering both hardware and software aspects.

Several open-source projects have emerged, focusing on hardware-based [RoT](#) implementations that enhance the security of embedded systems against fault injection attacks. These projects provide valuable resources for studying and improving fault-resilient designs. Notable examples include [OpenTitan](#) by LowRISC [[Low24](#)], a secure microcontroller designed to provide a robust foundation for hardware-based security; [CV32E40S](#) [[Ope24b](#)] powered by OpenHW group, a processor designed with built-in security features specifically targeting fault injection attacks; and [Caliptra](#) developed by ChipAlliance, a secure microcontroller currently under development and still in its early stages.

### 2.4.1 CPU Analysis against Fault Attacks

To assess the security of a [CPU](#) implementation, effective modeling and analysis methods are needed to gain a comprehensive understanding of fault effects [[YSW18](#)] while considering both hardware [[Arr+20](#); [Bur+17](#); [Ric+21](#)] and software [[DBP23](#); [HSP20](#); [Pot+14](#)]. These techniques are essential for uncovering microarchitectural details that can affect the system's overall security. Achieving this

requires automatically generated models that integrate hardware, software, and fault effects, and which can be processed by verification tools to enable automated verification.

To evaluate a CPU implementation, Tollec et al. [Tol+23] introduced  $\mu$ ArchiFI, an open-source tool designed for the formal modeling and verification of microarchitecture-level fault injections and their effects on complex hardware/ software systems. This approach allows for the analysis of fault effects across both hardware and software, offering a deeper understanding of system vulnerabilities.

Their methodology involves modeling faults at the microarchitectural level to accurately evaluate their impact on hardware and subsequent effects on software. They employ bounded model checking to reason about the consequences of fault injections and their potential exploitation by attackers. This technique has been applied to RISC-V use cases using state-of-the-art model-checking tools for hardware verification. The work was carried out on the CV32E40P processor, which is a processor without security features against hardware attacks.

### 2.4.2 Secure-oriented CPU Analysis against Fault Attacks

Building upon this work, during Silva Araújo's internship [Sil24] in collaboration with CEA/List, we adapted this tool to assess the security of the CV32E40S implementation. This adaptation enabled us to evaluate the processor's resilience against fault injection attacks, taking into account its specific architectural features and integrated countermeasures.

However, during this internship, several bugs were discovered in the CV32E40S implementation, which hindered our progress. As a result, we were unable to study the implementation of countermeasures based on fault models from the literature. Despite these setbacks, we were able to define and implement simple test programs, some of which were inspired by public implementations, such as the VerifyPIN from the FISCC project [Dur+16].

### 2.4.3 Synthesis and Perspectives

In this section, I present our work on analyzing protections against fault injection attacks in CPUs embedded in SE. This effort began during Silva Araújo's internship, where we adapted  $\mu$ ArchiFI to assess the security of the CV32E40S. To evaluate the processor's robustness, we used simple test programs, such as the VerifyPIN from the FISCC project [Dur+16].

This work represents an initial step in studying the security of processors designed to resist fault injection attacks. It also serves as a foundation for Alle Monne's Ph.D. thesis [All27], which focuses on evaluating the effectiveness of embedded countermeasures in the CV32E40S. His work will be a crucial step toward examining the security of application processors and their resilience to fault injection attacks.

Additionally, during Bikou's internship [Bik24], alongside Alle Monne's work [All27], we conducted a security analysis of OpenTitan. Its implementation is close to a commercial SE. However, unlike SE commercial solutions, OpenTitan has not undergone formal security evaluation and does not meet the industrial standards.

This analysis has provided valuable insights into the embedded security mechanisms within **SE**, which are crucial for securing **TEEs**.

## 2.5 Conclusion and Perspectives

This chapter summarizes my contributions to hardware **RoT** security. Building upon the research initiated during my **Ph.D. thesis** [Bou14], I expanded the investigation into the security of the Java Card platform, primarily embedded in **SE**. Additionally, we examined the security of communication interfaces, uncovering behaviors that could potentially fingerprint software implementations embedded in **SE**. Furthermore, I analyzed the security of the embedded **CPU** within **SE** against hardware attacks, with a particular focus on fault injection attacks, which are a significant threat to system integrity and confidentiality.

For future work, I will analyze the countermeasures integrated into hardware **RoTs** in order to resist hardware attacks, with the aim of understanding and evaluating their relevance and coverage. Subsequently, I will explore how to adapt these protections to application processors, securing them against hardware attacks and enhancing the security of **TEEs**. This work had begun under the supervision of **Alle Monne**'s Ph.D. thesis [All27].

## Chapter 3

# Contributions to the Trusted Execution Environment Security

As outlined in [Chapter 1](#), the [TEE](#) acts as the second layer of the [CoT](#), offering a secure environment for high-performance and sensitive applications. In 2015, [ANSSI](#) certified the [TEE protection profile](#) [[Glo20b](#)]. According to this [protection profile](#), evaluated [TEE](#) implementations should be protected against software and hardware attacks. Despite the complex architectures of application [CPUs](#) that integrate [TEEs](#), hardware attacks have become increasingly sophisticated and feasible. This mirrors the evolution of attacks aimed at hardware [RoTs](#) [[MBB16](#); [TSW16](#); [Vas+20](#)]. The large attack surface of application [CPUs](#) makes securing [TEEs](#) increasingly challenging [[TM17](#); [YSW18](#)].

In 2016, an evaluation of [OS](#) security within [TEEs](#) was initiated in France under the Common Criteria scheme, supervised by [ANSSI](#) and aligned with the [TEE protection profile](#) [[Glo20b](#)]. Although comprehensive, the assessment of hardware attacks, such as fault injection, was out of scope due to insufficient knowledge at the time to fully evaluate the necessary conditions for successful exploitation by attackers.

It is in this context that my research on [TEE](#) security began in early 2017, as evidence of successful fault injection attacks against [TEEs](#) grew [[MBB16](#); [TSW16](#); [Vas+17](#)]. These attacks often replicated those previously applied to [SEs](#) without considering the specific characteristics of the application [CPUs](#) running [TEEs](#).

This chapter is organized as follows. First, [Section 3.1](#) presents the common architecture of a [TEE](#), where segregation between the [REE](#) and [TEE](#) is based on both hardware and software. Next, [Section 3.2](#) discusses the hardware architecture of application [CPUs](#) running [TEEs](#), often embedded in a [SoC](#). While these application [CPUs](#) coexist with many modules, they are not designed to resist hardware attacks. Then, [Section 3.3](#) addresses the risks of hardware attacks on application [CPUs](#) and introduces my contributions to characterize resistance to fault injection attacks from both software and hardware perspectives, with the aim of designing countermeasures to protect [TEEs](#). [Section 3.4](#) follows

by discussing the risks introduced by integrating application CPUs running TEEs into modern SoCs shared with various modules. These modules, accessible from the REE, can be exploited to compromise the confidentiality and integrity of TEEs, highlighting the need for high-level countermeasures against hardware attacks from these modules. Finally, Section 3.5 concludes the chapter and outlines future perspectives.

### 3.1 Common Trusted Execution Environment Architecture

A TEE is a software enclave within the main application CPU, designed to provide a secure area. The most widespread implementation of TEE is that of Arm, which represents 99% of the deployed mobile CPUs [Kin24]. In Arm's TEE implementation, named TrustZone, both the REE and TEE run on the same CPU. The TEE software typically runs on a (generally fixed) core and has access to specific peripherals used to perform sensitive operations. Upon a hardware scheduling event, the core switches from the REE to the TEE, isolating the execution of trusted applications from the rest of the system. Unlike software implementations embedded in SE, most parts of the TEE software architecture are standardized, with many specifications provided by GlobalPlatform working groups [Glo20a; Glo20b; Glo21]. Figure 3.1 shows a common TEE software architecture along with these specifications.

Within the TEE, trusted applications run in a protected environment with limited access to the outside world, operating independently from the REE. Communication between the TEE and external environments is handled through well-defined interfaces, setup by the secure monitor [Glo20b], managed by the TEE OS. The TEE OS enforces security policies and access controls, ensuring that only authorized REE applications and processes can interact with the TEE. Additionally, the TEE OS ensures that sensitive operations are executed in a secure manner. The integrity of the TEE OS is verified by a RoT during the secure boot process, while the overall security of the TEE runtime environment is maintained by the TEE OS.

As shown in Figure 3.1, the isolation between the REE and the TEE worlds is enforced by the underlying hardware architecture. Consequently, the security of the hardware layout is crucial to maintaining the integrity and security of the TEE environment. Application CPUs, on which TEEs run, are primarily designed with performance in mind rather than security, and thus often lack inherent protections against hardware attacks.

In the mid-2010s, several hardware attacks transposed from those initially targeting SEs were successfully carried out on application CPUs running TEE [Bal+15; MBB16; TSW16; Vas+17; Yan+15]. Although these attacks were not specifically adapted to the architecture features of the targets, they demonstrated that application CPUs are susceptible to hardware attacks and that attackers can exploit this vulnerability. Towards the end of the 2010s, further research showed that the particular characteristics of application CPUs could also be leveraged to compromise the security of TEE applications [Koc+19; Lip+18; Man+18; TSS17].

To understand the consequences of hardware attacks on TEEs, my research focuses on analyzing

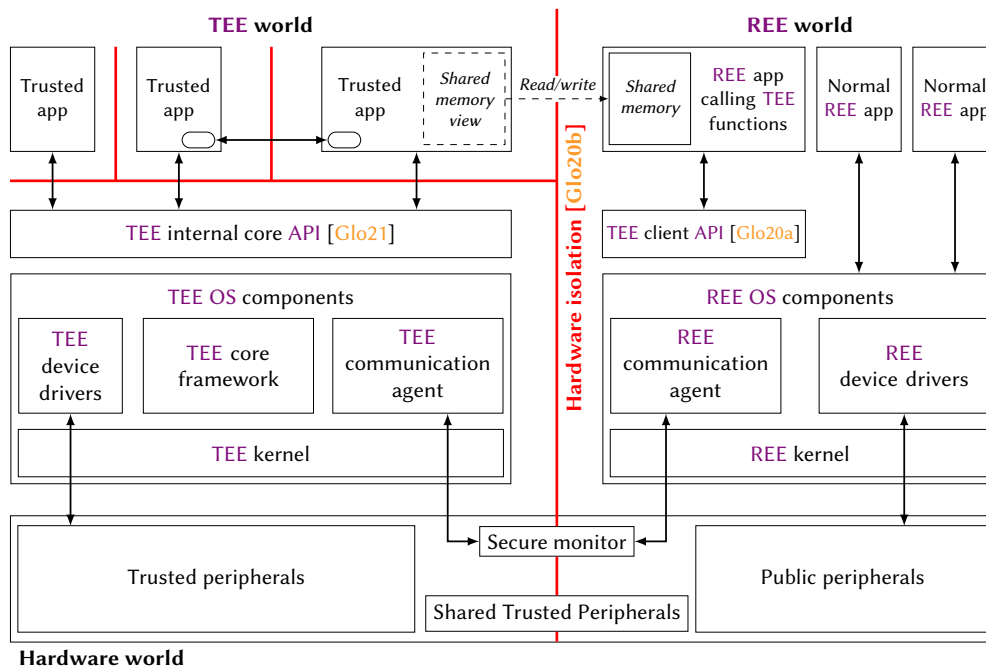


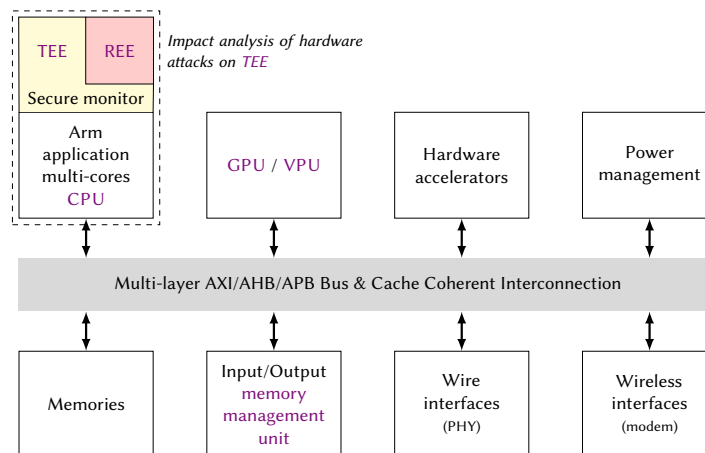
Figure 3.1 – Common TEE software architecture. Adapted from [Glo20b].

their effects and potential when targeting application CPUs. Since Arm dominates the embedded systems market [Kin24], I will focus on this implementation in my study. However, while the following works specifically target COTS component Arm application processors, the proposed approaches are agnostic and can be applied to other processor implementations running a TEE.

## 3.2 Hardware Architecture of Application Processors

Unlike the architecture of SEs, described in Figure 2.1, which features a CPU with built-in fault detection mechanisms embedded within a minimally functional SoC to maximize security, application CPUs are integrated into complex SoCs that incorporate numerous modules; such as Graphics Processing Units (GPUs), Video Processing Units (VPUs) with artificial intelligence engines, and power management units for optimizing energy efficiency. Figure 3.2 illustrates the hardware and software architecture of such an application SoC embedding an Arm processor.

To enhance software execution, application SoCs utilize internal memory called cache to store copies of data from external memory. Caches are organized into a hierarchy of levels, each differing in size, speed, and proximity to the CPU cores. In multi-core CPUs, each core has its own dedicated Level 1 (L1) cache, the smallest and fastest, allowing fast access to frequently used data and instructions. Level 2 (L2) caches are larger and may be private to each core or shared among cores, serving as intermediaries between the L1 caches and external memory. Level 3 (L3) caches are even larger and generally shared



**Figure 3.2** – Overview of the hardware and software architecture of a modern application SoC embedding an Arm CPU running a TEE. Arm is the most deployed architecture featuring a TEE, known as TrustZone [Kin24]. This technology provides a hardware-based security mechanism [Arm17], where the TEE and REE run on the same CPU. Adapted from [BRS17].

among all cores, simplifying efficient data sharing and maintaining coherence across the processor. This hierarchical cache structure reduces memory access latency and enhances overall system performance by keeping data closer to the CPU cores.

Application CPUs feature superscalar pipelines for fetching and decoding multiple instructions in parallel, and often include out-of-order execution stages, which improve performance by allowing instructions to be executed in an order that maximizes efficiency, rather than strictly following the original program sequence. However, these performance enhancements add complexity to the CPU architecture, which can introduce new security challenges and increase the attack surface for hardware attacks [Koc+19; Lip+18].

These CPUs incorporate a Memory Management Unit (MMU) that translates the virtual addresses used by software processes into physical addresses. The MMU plays a crucial role in managing memory protection and access control, ensuring that processes access only the memory regions they are permitted to. Additionally, modern CPUs integrate hardware security mechanisms such as Control Flow Integrity (CFI) and pointer protection techniques like Pointer Authentication Code (PAC) or Capability Hardware Enhanced RISC Instructions (CHERI) [Woo+14] to mitigate software attacks by restricting control flow manipulations and preventing memory safety violations. While these mechanisms provide strong protections against software attacks, they are not designed to withstand hardware attacks.

While this complexity is essential for meeting the requirements of performance-intensive applications, it also significantly increases the attack surface, often relegating security to a secondary consideration. The intricate design and the multitude of integrated modules introduce additional vulnerabilities that can be exploited by attackers [Cam+18]. Consequently, application processors become more susceptible to hardware attacks that can compromise the entire system's security.

In the rest of this manuscript, I refer to micro-architecture blocks as each individual component within a CPU's architecture, such as caches, pipeline stages, branch predictors, and execution units, which contribute to the overall performance and functionality of the system [Wik24c].

### 3.3 Impact of Hardware Attacks on High-Performance Processors

#### Associated contextual elements of this research field

<b>Collaborations:</b>	Jessy Clédière (CEA/Leti, Grenoble, France), Rachid Dafali (DGA-MI, Bruz, France), Ronan Lashermes (INRIA/LHS, Rennes, France), and Olivier Sentieys (INRIA/TARAN, Rennes, France).
<b>Publications:</b>	[Mar+24; TBC19; TBC21; Tro+21]
<b>Supervisions:</b>	Thomas Troughkin [Tro21] and Amélie Marotta [Mar25] Ph.D. theses.

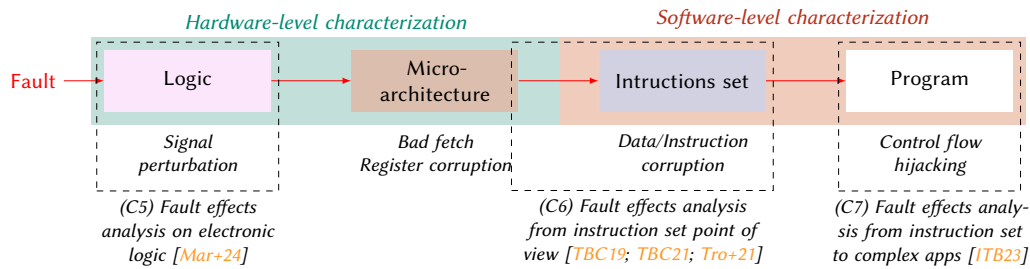
Despite the architecture complexity highlighted in Figure 3.2, the practical feasibility of hardware attacks on application processors has been a topic of debate within the scope of security evaluation schemes. Hardware attacks can be categorized into two main types: passive attacks (or observation attacks), known as side-channel attacks, where sensitive information is extracted through physical observations during execution, and active attacks (or perturbation attacks), referred to as fault injection attacks, where the attacker deliberately disrupts the system's behavior to induce computation errors at runtime, potentially leading to the disclosure of sensitive information or unauthorized access. Attacking a application CPU typically involves targeting it through the SoC in which it is embedded.

To strengthen the security of high-performance software and hardware implementations, it is essential to understand the potential risks posed by hardware attacks to develop secure countermeasures accordingly.

In the context of side-channel attacks, prior research has already explored their feasibility and impact [Bal+15; Lon+15] on application CPUs. However, when I began this research in 2017, the exploitability of fault injection attacks on these architectures remained uncertain and was a subject of ongoing discussion. To address this gap, my research focused on investigating methods to characterize the effects of such attacks on application CPUs, as summarized in Figure 3.3.

Unlike fault injection attacks on security-oriented microcontrollers, where the system architecture is relatively straightforward, the intricate nature of application CPUs significantly complicates the analysis and interpretation of the resulting faults. In scenarios where the target CPU is a COTS component and its implementation details are not available, the analysis must be conducted from the ISA level [Pro+19]. This approach allows us to observe the behavior of the CPU exclusively through its ISA interface, enabling the evaluation of fault impact solely based on observed behavior, without direct access to the underlying hardware design. My contributions to this area are described in Section 3.3.1.

When the design of an application CPU is accessible, it is possible to gain a deeper understanding of fault effects at the electronic logic level. Such an in-depth analysis can be carried out at various



**Figure 3.3** – Fault propagation through digital device abstraction layers, adapted from [YSW18]. Levels of characterization (hardware or software) are indicated above the perturbed blocks, and the examples of fault effects are shown below. Dashed boxes and italicized labels, started with the (C5), (C6) and (C7), indicate my contributions to fault injection at hardware and software level.

abstraction layers by simulation, including the **Register Transfer Level (RTL)** [Tol+24], the netlist [Ric+21], and even down to the mask level [Zho+22]. Physical experimentation through prototyping on **Field-Programmable Gate Arrays (FPGAs)** [Mar+24] or silicon implementations on **Application-Specific Integrated Circuits (ASICs)** [Bar+06] also provides valuable insights. Each of these layers offers a distinct perspective on how faults propagate within the CPU’s architecture, which helps refine the evaluation of potential vulnerabilities. My work in this area is detailed in **Section 3.3.2**.

### 3.3.1 Fault Effects Characterization from the Software to the Hardware

#### Related publications

- 📄 Thomas Troughkine, Guillaume Bouffard, and Jessy Clédière. “Fault Injection Characterization on Modern CPUs”, In: *WISTP 2019* [Article in PDF](#) [TBC19] 🏆 Best student paper award
- 📄 Thomas Troughkine, Guillaume Bouffard, and Jessy Clédière. “EM Fault Model Characterization on SoCs: From Different Architectures to the Same Fault Model”, In: *FDTC 2021* [Article in PDF](#) [TBC21]
- 📄 Thomas Troughkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, and Guillaume Bouffard. “Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models”, In: *Journal of Cryptographic Engineering* 2021. [Article in PDF](#) [Tro+21]

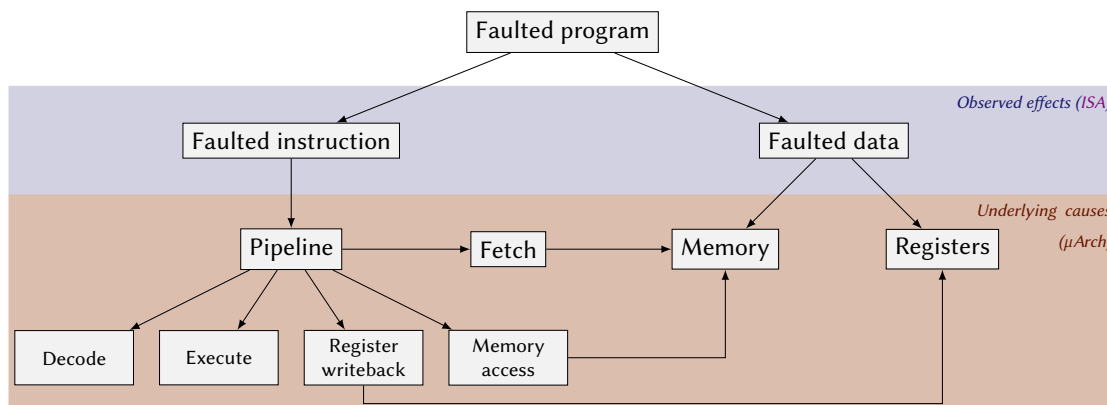
When evaluating the security of software against fault injection attacks on a application **COTS component**, it is necessary to understand the impact of such attacks on software execution without knowledge of the underlying hardware implementation. To conduct an in-depth study of how hardware perturbations alter software behavior, Proy et al. [Pro+19] proposed analyzing the effects of faults on the execution of simple programs written in assembly language. By strategically selecting instructions to execute during fault injections, we can observe how faults propagate within the software. This approach provides a foundational understanding of how faults affect software behavior, starting from isolated instructions and extending to more complex program structures. While this method offers valuable insights, it has certain limitations, particularly the lack of a generalizable approach in designing the code samples, which may limit its applicability across different architectures.

To address these limitations, we generalized Proy et al.’s method [TBC19] to support different architectures [TBC21]. This generalization allowed us to better understand and characterize the effects of fault injection attacks across a wider range of application CPUs, enhancing the applicability and robustness of the analysis.

In our approach, we perform the analysis from the ISA level to determine the fault model for fault injection attacks. While multiple fault effects can occur due to the complex nature of application CPUs, it is impractical to characterize all possible micro-architecture blocks individually. We focus on the observable effects from the instruction set, which provides a practical abstraction of the underlying hardware complexities. This contribution is labeled (C6) in Figure 3.3.

### 3.3.1.1 Fault Effects Characterization: From ISA to Micro-Architecture Blocks

During a fault injection, one or more CPU micro-architecture blocks may be perturbed. While characterizing the effects across all possible blocks is complex, prior works on microcontrollers have shown that faults typically impact a single micro-architecture component [KH14; Riv+15]. We confirmed this behavior from the ISA point of view on in-order application CPUs [TBC19], thereby simplifying the problem to identify *which* micro-architecture block is affected and *how* [TBC19]. Our approach follows a top-down structure: starting by identifying whether the fault targets the data or the instructions, then narrowing down to specific blocks, as summarized in Figure 3.4.



**Figure 3.4** – Top-down approach to characterize fault effects on in-order application processors micro-architecture blocks.

To isolate the faulted micro-architecture block, we repeatedly execute carefully chosen instructions on a known-state CPU that (1) do not alter the processor state (any modification indicates a fault) and (2) do not involve memory access, thus minimizing the analysis scope. In [TBC19], we proposed a formal approach to define these instructions, enabling us to determine which micro-architecture block is perturbed during a fault injection.

By analyzing the distribution of faulted values, we determine whether the error originates in the data or the instructions. Depending on the classification depicted in Figure 3.2 and using the fault model

obtained from the faulted program, we can trace the fault to specific blocks. For example, a register corruption points directly to faulted data, while incorrect instruction fetching may indicate issues in memory access, either in the cache or external memory.

Building upon this approach, we extended our methodology to **COTS components** from different architectures, specifically **in-order** Arm and Intel processors. In our study [TBC21], we demonstrated that fault injections on these diverse architectures result in similar fault models from **ISA** point of view, validating the generality of our method. This cross-architecture applicability confirms that our approach is portable and can be utilized for security evaluations on a wide range of application **CPUs**.

Moreover, in collaborative work with Inria, we analyzed how fault injections can affect memory management and cache systems on **in-order** application **SoCs** [Tro+21]. This work shows how the **MMU** and cache mechanisms, both critical components responsible for virtual memory handling, access control, and efficient data access, can be compromised. We investigated how fault attacks can disrupt memory access operations, potentially leading to incorrect virtual-to-physical address translations and allowing unauthorized accesses to protected memory regions. Additionally, we studied the effects of faults on cache systems, revealing how perturbations can corrupt cached data or instructions, leading to persistent inconsistent system states. Understanding these fault effects is essential for developing more effective software-level countermeasures that ensure system security even when hardware faults occur.

### 3.3.1.2 Synthesis and Perspectives

During Troughkine's Ph.D. thesis [Tro21], we further improved and generalized the approach proposed by Proy et al. [Pro+19] by formalizing it [TBC19]. This formalization enabled its application across different architectures [TBC21] and extended it down to the micro-architecture blocks [Tro+21], providing finer-grained insights into which specific micro-architecture blocks are affected by faults, as observed from the **ISA** point of view. This refined methodology can help in developing more efficient software-level countermeasures tailored to address fault effects in specific **COTS components**. All the experiments conducted during Troughkine's Ph.D. thesis are available on [ANSSI-FR/Faults\\_experiments's GitHub](#), and the tool to analyze them is available at [ANSSI-FR/Faults\\_analyzer's GitHub](#). However, this approach represents only an initial step towards fully characterizing fault effects from the **ISA** point of view. The proposed approach currently covers only a subset of the entire **ISA**, focusing on memory accesses and register manipulations, and should be extended to comprehensively characterize all fault effects from the **ISA** point of view.

The fault characterization approach at the **ISA** level [TBC19] targets simple programs where the **CPU** state is easily known. However, application **CPUs** mainly run complex software targeted by fault attacks [TSW16], where the **CPU** state is very hard to determine. To detect potential fault on complex software applications from perturbation attacks, the approach introduced during Troughkine's Ph.D. thesis [Tro21] needs to be scaled.

In [Gai+20], Gaine et al. studied a complex binary application but modified the target code by inserting a trigger to ease fault injection at the correct time. In real-world scenarios, if an attacker

is able to modify the binary applications running on the target, they can perform software attacks directly, making hardware attacks unnecessary. In my research, hardware attacks are considered when the software layout cannot be instrumented or modified by an attacker. My goal is to extend our fault model to complex software without modifying the target application.

Fanjas et al. [Fan+22] improved upon the work of Gaine et al. [Gai+20] by combining side-channel attacks to detect the optimal moment to inject the fault. While this work is interesting, they only reproduced the fault found in [Gai+20]. Their approach is more of a bottom-up strategy, asking, “*We have a sensitive fragment of code, how do we break it?*”. In contrast, the top-down approach I wish to adopt is: “*Given how our hardware target is sensitive to fault attacks, which application paths can be corrupted to obtain sensitive assets?*”.

In [ITB23], we presented our approach to extend a fault model from the ISA level to complex software applications. This contribution is labeled (C7) in Figure 3.3.

To address this problem, we studied complex programs such as `sudo`<sup>1</sup>, which dynamically load several shared libraries to provide the appropriate authentication methods.

To successfully perform a perturbation attack on such complex software, it is first necessary to determine the fault models using an open sample of the same component model that provides development access. Then, by performing static code analysis on the target binary application, using a fault injection simulator like Rainbow<sup>2</sup>, we can identify the sensitive execution paths that are critical for security. This is feasible because firmware binaries are generally available on the Internet or can be extracted by dumping the target’s memory.

Finally, the exploitation phase involves applying the obtained fault model at the precise moment during execution. This work is ongoing. We have obtained some promising initial results, but further research is needed to better understand the challenges posed by complex software environments.

### 3.3.2 Fault Effects Characterization from the Hardware to the Software

#### Related publication

📖 Amélie Marotta, Ronan Lashermes, Guillaume Bouffard, Olivier Sentieys, and Rachid Dafali. “Characterizing and Modeling Synchronous Clock-Glitch Fault Injection”, In: *COSADE 2024* [Article in PDF](#) [Mar+24]

Section 3.3.1 introduces my study on understanding the effects of perturbation attacks at the ISA level, focusing on how faults manifest in software execution without detailed knowledge of the underlying hardware implementation. However, when we have access to the hardware implementation details, we can go deeper into how the hardware itself resists to perturbation attacks. This allows for a more

1. `sudo` (short for “*superuser do*”) is a command-line utility in Unix and Linux-based OS that allows a permitted user to execute commands with the privileges of another user, typically the superuser (root). It enables administrative tasks to be performed securely without granting full root access to users.

2. Rainbow is an open-source fault injection simulator developed by Ledger Donjon, based on Unicorn-Engine (QEMU). Rainbow is available on [Rainbow’s GitHub](#)

comprehensive analysis of the system’s fault tolerance and the identification of vulnerabilities at the hardware level.

To understand fault injection attacks on hardware where the implementation details are known, we can perform simulations from the RTL level to the mask level. Deploying the implementation on FPGAs or ASICs enables practical experimentation, complements the results obtained in simulation, and validates the implementation’s resistance to fault injection attacks. Studying the security of the implementation in its final form, whether as an FPGA or an ASIC, is particularly pertinent because it reflects the actual hardware configuration that will be deployed, including any physical characteristics that may influence fault susceptibility. However, due to the complexity of an application CPU, analyzing it in its entirety is a challenging task. It is difficult to understand how a signal perturbation at the logic level can translate into erroneous behavior at the micro-architecture level, as depicted in the left part of Figure 3.3.

To address this challenge, we began by measuring the effects of fault injection attacks on component logic. During Marotta’s Ph.D. thesis [Mar25], we focused on flip-flops, which are crucial components in computer electronics. Flip-flops function as memory elements that store state information. They ensure clock synchronization, support digital counting, and contribute to control logic. Understanding how flip-flops behave under fault injection is essential because their disruption can significantly affect the overall operation of the CPU. This contribution is labeled (C5) in Figure 3.3.

### 3.3.2.1 Fault Effects Characterization: Fault Effects on Logic

Measuring the behavior of flip-flops under fault injection is crucial, as they play a key role in maintaining state information in digital circuits. In [Mar+24], we focused on the impact of clock glitches on flip-flops, given that the clock signal is their most sensitive input. At each clock edge, the flip-flop evaluates its inputs and updates its output, making any disruption to the clock signal particularly impactful on its proper functioning. Clock glitches can be induced either by Electromagnetic Fault Injection (EMFI) [CB19] or by an attacker directly manipulating the clock signal.

To evaluate how flip-flops react when the clock signal is altered, we used TRAITOR [Cla+21] to generate controlled clock modifications. TRAITOR can precisely adjust the amplitude parameter, which defines the energy level of the synchronous clock glitch. This capability enables a more accurate control of the glitch, referred to as the controlled synchronous clock glitch. In our setup, a set of flip-flops was located on an FPGA board where the clock was managed by TRAITOR.

Our experiments revealed that when the energy of the clock signal falls below a critical threshold, flip-flops fail to sample the input correctly, leading to faults. Existing fault models (such as the timing fault model [Ago+10; SGD08], the sampling fault model [DLM21], and the charge-based fault model [LG19]) attempt to explain these effects, but none fully capture the behavior we observed. To address this gap, we proposed a new fault model: the *Energy-Threshold Fault Model*, where the energy delivered by the clock’s rising edge determines the fault occurrence. This fault model is defined by three distinct behaviors. In the *Always faulted* case, the clock signal’s energy is too low, leading to consistent faults. In the *Sometimes faulted* case, the clock energy hovers around the threshold, resulting in metastable

states. Finally, in the *Never faulted* case, the clock signal's energy exceeds the threshold, allowing normal operation.

To further comprehend the observed fault model, we conducted simulations by varying both the clock pulse width and voltage amplitude independently. The results demonstrated that voltage amplitude plays a more significant role than the width of the glitch in determining whether the flip-flop samples correctly. Specifically, a higher voltage amplitude with a short pulse width is sufficient for proper sampling, while a lower voltage, regardless of width, fails to do so.

Additionally, we explored why different flip-flops exhibit varying fault sensitivities. Our hypothesis suggested that intrinsic factors, such as manufacturing variability and clock routing, could influence the fault sensitivity of individual flip-flops. However, further investigation revealed that extrinsic factors, particularly the activity on neighboring data and clock lines, could also impact fault sensitivity due to signal cross-talk. For example, placing a control clock signal adjacent to the glitched clock signal on the same **FPGA** slice reduced the fault sensitivity by increasing the energy of the glitched clock through cross-talk.

Combining findings from both physical experiments and transistor-level simulations, we concluded that fault sensitivity is influenced by both intrinsic properties of the flip-flops and extrinsic factors from their surrounding circuitry. This characterization of fault behavior in flip-flops under synchronous clock glitch conditions provides a deeper understanding of fault injection vulnerabilities in synchronous digital circuits and supports the development of more resilient countermeasures.

### 3.3.2.2 Synthesis and Perspectives

In this section, as presented in **Marotta**'s Ph.D. thesis [**Mar25**], we studied the effects of perturbation attacks on the logic of a component whose implementation details are known. Moving forward, it is necessary to bridge the gap between observations made at the logic level and those at the **ISA** level introduced in **Section 3.3.1**, thereby understanding the consequences on the implementation of micro-architecture blocks.

While approaches exist that allow security analysis at the **RTL** level, such as  **$\mu$ ArchiFI** [**Tol+23**], there are multiple steps between the **RTL** and the mask used to fabricate an **ASIC** or the bitstream for a **FPGA**. To protect the security of the software execution environment against hardware attacks, it is essential to understand the different design steps and how security properties are preserved between **RTL** design and synthesis. How are security properties preserved at each of these stages? How can we ensure that security is maintained throughout the design flow?

### 3.4 Impact of Environment on Application Processors Security

#### Associated contextual elements of this research field

- Collaboration:** Maria Méndez Real (Université de Bretagne Sud, Lab-STICC, Lorient, France) and Jean-Christophe Prévotet (IETR & INSA, Rennes, France).
- Publication:** [Gon+25]
- Supervision:** Gwenn Le Gonidec’s Ph.D. thesis [Gon26].

#### Related publication

📄 Gwenn Le Gonidec, Guillaume Bouffard, Jean-Christophe Prévotet, and Maria Méndez Real. “Do Not Trust Power Management: A Survey on Internal Energy-based Attacks Circumventing Trusted Execution Environments Security Properties”, In: *ACM Transactions on Embedded Computing Systems* 2025 [Article in PDF](#) [Gon+25]

When a **SE** is embedded in the same **SoC** as an application **CPU**, it must comply with a specific **protection profile** [Eur22] that takes into account the particularities of the hardware runtime environment. In such environments, numerous integrated modules could potentially be exploited maliciously, even if they were not originally designed with malicious intent, to compromise the embedded **SE**. As shown in [Figure 3.2](#), modern **SoCs** integrate many modules, some of which can be exploited, from a compromised **REE**, to undermine the security of **TEE** applications [Cam+18; TSS17].

State-of-the-art attacks have demonstrated that certain **SoC** modules can be leveraged to corrupt or extract sensitive data used by **TEEs**. For example, wireless modems have been used to extract secret keys [Cam+18]. Another frequently targeted module is the **Power Management Unit (PMU)**. Modern **SoCs** architectures integrate **PMUs** to optimize energy efficiency and extend battery life. While **PMUs** are crucial for controlling voltage and frequency scaling, they can also serve as vectors for hardware attacks [Gon+25]. Unlike traditional hardware attacks that require physical access to the device, adversaries can potentially exploit **PMUs** remotely, to perform malicious activities [TSS17], such as voltage glitching, which can compromise both **TEEs** and the entire system. This remote exploit significantly heightens the threat level, as attacks can be conducted without any physical interaction with the target device.

As part of Gonidec’s Ph.D. thesis, which began in late 2023 [Gon26], we are examining the implications of integrated **PMUs** on the security of **TEEs**. By studying how these modules can be manipulated to induce faults or leak sensitive information, we aim to identify vulnerabilities and propose mitigation strategies. As a first step, we analyzed the state of the art [Gon+25] to highlight how **PMUs** pose a significant risk to **TEE** security. We found that the proposed countermeasures are not well adapted to protect the **TEE** environment, mainly because they follow a bottom-up approach that focuses mainly on published attacks and often introduces significant runtime overhead, which reduces the feasibility of executing sensitive operations in application environments.

For the next phase of Gonidec’s Ph.D. thesis [Gon26], we will focus on characterizing the effects of

fault injection attacks originating from the **PMU** and how this hardware module can compromise the **TEE** environment. This ongoing work aims to understand the capabilities of an attacker exploiting the **PMU** to perform hardware attacks. Based on this understanding, we will explore high-level countermeasures to protect the **TEE** from fault injection attacks facilitated by the **PMU**.

Although this research initially targets the assessment of the consequences, feasibility, and mitigations of hardware attacks via power management modules against **TEEs**, the proposed approach can be extended to all hardware blocks embedded in an application **SoC**. This includes hardware **RoTs**, **REE**, and cryptographic accelerators, which may also be vulnerable to similar attack vectors.

### 3.5 Conclusion and Perspectives

This chapter explores the security challenges faced by **TEEs** when subjected to hardware attacks, particularly fault injection attacks targeting application **CPUs**. Through our investigations at both the **ISA** level and the electronic logic level, we aimed to understand the effects of fault injections and how they propagate through the system. However, a disconnection exists between the fault effects characterization at the **ISA** level and one at the electronic logic level. By linking these observations at different levels of abstraction, I aim to understand how fault injection attacks affecting micro-architecture blocks can corrupt software execution. This understanding is needed to design effective countermeasures to protect **TEEs** against such attacks.

The integration of application **CPUs** running **TEE** into modern **SoCs**, which are shared with numerous modules, introduces new risks. These modules, accessible from the **REE**, can be exploited to breach the confidentiality and integrity of sensitive data in **TEEs** applications. For example, integrated modules like **PMUs** present potential attack vectors that adversaries can leverage to compromise the **TEE** environment. Remote manipulation of these modules from a compromised **REE**, such as inducing faults via voltage glitches, poses a significant security challenge. Understanding these risks enables us to develop high-level countermeasures tailored to this environment.

While my contributions have predominantly focused on fault injection attacks, it is imperative to also consider side-channel attacks to comprehensively secure **TEEs**. Side-channel attacks exploit information leakage through physical phenomena such as power consumption, electromagnetic emanations, or timing variations. By broadening the scope to include both fault injection and side-channel attacks, I aim to develop a more robust security framework for **TEEs** in high-performance computing environments.



## Chapter 4

# Contributions to Execute Sensitive Applications in the Rich Execution Environment

As introduced in [Chapter 1](#), the [REE](#) is the last layer of the [CoT](#), providing a rich and flexible execution environment for applications requiring high performance. This layer is designed to execute application *business logic without built-in security mechanisms*, whereas the [TEE](#), as illustrated in [Figure 1.2](#), is specifically designed to execute *business security functions*.

To offer complementary applications addressing user needs, such as gaming, video streaming, or online shopping, third-party developers primarily target high-performance devices, such as personal computers, tablets or smartphones. These applications are designed to run on devices owned and controlled by end users, leveraging the performance and flexibility of [COTS component](#). In this context, the [REE](#) represents an execution environment under the user's control, which is therefore considered potentially untrusted from a security standpoint. Third-party applications are generally developed without prior knowledge of the embedded features available within the [REE](#).

As discussed in [Chapters 1](#) and [3](#), the use of the [TEE](#) is recommended to improve the security of applications running in the [REE](#). However, their usage remains at the discretion of the platform owner, as access to [TEE](#) is exclusively governed by device manufacturers and requires specific agreements. Given the wide variety of manufacturers, establishing such agreements is complex, further complicating the integration of security mechanisms for third-party applications.

Moreover, the presence and availability of such security features are not directly accessible to applications. It is the [REE OS](#) that advertises the available security features of the platform, such as the presence of a [TEE](#) or a hardware [RoT](#). This implies that applications running in the [REE](#) rely on the [OS](#) to obtain information about the underlying security capabilities.

This chapter will present my research activities related to the execution of sensitive applications in an untrusted execution environment. Although relying on the TEE is generally considered a recommended approach to secure such applications, the lack of agreements makes it difficult to obtain access across all target devices. Furthermore, as discussed in Figure 3.1, all communications between REE applications and the TEE transit through the REE OS, which is considered untrusted by the TEE security model. This raises concerns about the possibility of an OS that could mislead applications by eavesdropping on interactions or lying about the actual capabilities supported.

The remainder of this chapter is organized as follows. Section 4.1 introduces the architecture and operating principles of a REE, highlighting its interactions with the TEE and the hardware RoT. Section 4.2 then presents my research activities related to the design of an obfuscated application intended to execute within a REE. In this work, we considered an attacker in a white-box model [Cho+02], capable of instrumenting the application in order to extract its secrets. To model this threat, we envisioned an adversary able to transpose hardware attacks onto the application. Finally, Section 4.4 concludes this chapter by discussing some perspectives.

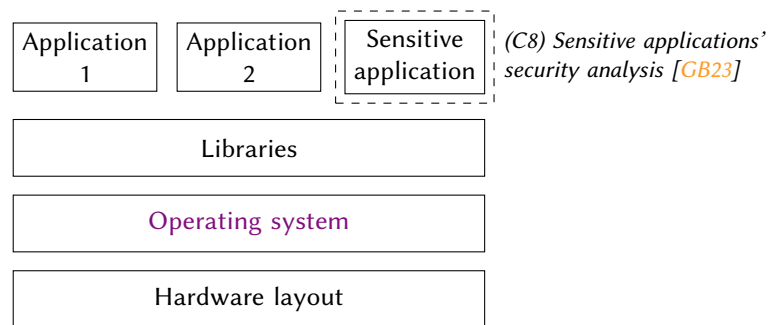
## 4.1 Common Rich Execution Environment Architecture

A REE represents the runtime environment in which application software is executed on consumer devices, such as smartphones, tablets, and personal computers. It provides a rich set of functionalities, including support for graphical user interfaces, network communications, and access to various hardware peripherals, enabling the execution of performance-demanding applications. Unlike the execution environments of the TEE and the hardware RoT, which are designed for isolated and trusted operations, the REE is a multi-application environment where a user, granted a specific set of privileges, can install and run applications coded by different developers. These applications may be installed from sources with varying levels of trust, such as official application stores or unverified website from the Internet. To manage interactions between these applications and ensure the separation of user data and processes, the system enforces access control policies.

However, this flexibility comes with a lower security assurance compared to environments like the TEE, mainly due to the larger attack surface exposed by the complexity and openness of the REE. Figure 4.1 illustrates a typical REE architecture as integrated into modern execution environments.

At the bottom layer of Figure 4.1, the *Hardware layout* represents the physical components of the platform, including the CPU, memory, storage, and peripherals (e.g., sensors, cameras, and wireless interfaces). The hardware provides the foundational resources required for the execution of the OS and applications.

Above the *Hardware layout*, the *Operating system* is responsible for abstracting the hardware complexity and offering standardized services to upper layers. In a REE, the OS typically ensures the isolation of applications through process and memory management mechanisms, based on hardware MMU,



**Figure 4.1** – A common REE architecture inspired from [Nat21a]. Dashed box and italicized label, started with the (C8), indicate my contribution to security of running sensitive applications in an untrusted REE.

prevents unauthorized access to system resources, and supports a multi-user environment by assigning privileges and enforcing access control policies. It also provides device drivers to interface with hardware components, enabling secure and efficient peripheral management. Additionally, the OS advertises the presence of security capabilities such as the TEE or hardware RoT to applications. Finally, it facilitates inter-process communication, allowing applications and system services to exchange data.

The *Libraries* layer includes software components shared across applications, such as graphics libraries, cryptographic primitives, and networking stacks. Libraries simplify application development by offering high-level abstractions and reducing redundant code.

At the top layer, *Applications* represent third-party or native software executed within the REE. These applications leverage the OS and libraries to access system resources and deliver functionalities to end users. Examples include messaging apps, video streaming platforms, or online shopping applications. Applications in the REE are typically installed from an application store embedded in the device, or downloaded from the Internet when users are allowed to install software from external sources. They generally operate under the assumption that the OS maintains its integrity and correctly enforces its security policies, although it is treated as potentially malicious from the perspective of secure execution environments like the TEE.

This layered view emphasizes the reliance of applications on the OS for hardware access and security features. It also highlights the potential risks associated with the REE's large attack surface, given the complexity and openness of this environment.

## 4.2 Software Security in the Rich Execution Environment

### Associated contextual elements of this research field

**Collaboration:** David Naccache (ENS, Paris, France)

**Publication:** [GB23]

**Supervision:** Vincent Giraud's Ph.D. thesis [Gir24].

The **REE** provides a flexible and feature-rich environment running on an application processor that embeds hardware security mechanisms designed to enhance software protection, such as **CFI**, **PAC**, and **CHERI** [Woo+14]. While these mechanisms help mitigate certain classes of attacks, the **REE** remains an open environment designed to run applications developed by various third parties. Sensitive applications often execute in an environment that has not undergone a full security evaluation or has only undergone a low-assurance assessment on a few specific features [Nat21b], further increasing the risk of exposure to threats from unevaluated components. This raises fundamental questions regarding their security, particularly when sensitive applications are executed within the **REE**.

Applications handling sensitive data, such as banking, healthcare, or digital identity services, rely on the **REE OS** to mediate access to hardware resources and security services. In security evaluations, the **OS** is generally assumed to be authentic and unmodified [Nat17; Nat21a]. Users are also encouraged to maintain the device security by keeping their **OS** up to date, avoiding modifications such as rooting or jailbreaking, and following security best practices [Nat17; Pay19].

However, the assumption that the **OS** remains up to date and that neither the user nor a malicious application alters its integrity does not always hold in practice. In the case of mobile **OS**, updates are not guaranteed indefinitely. For instance, Google smartphones typically receive security updates for a maximum of seven years [Bro24], leaving older devices vulnerable to unpatched security flaws. Furthermore, if an attacker, whether the user or a malicious application, successfully gains control over the **OS**, they can conceal this modification using tools such as **Magisk**, making it difficult for sensitive applications to reliably assess its trustworthiness.

As a result, the **REE OS** is generally considered potentially untrusted since it could interfere with the execution of applications, eavesdrop on sensitive data, or manipulate security-critical operations. This threat model challenges the assumption of software security in the **REE** and underscores the need for additional protection layers, such as leveraging the **TEE** or obfuscation techniques. Nevertheless, access to the **TEE** is not always available to third-party developers due to the need for prior agreements with the platform provider, further complicating the adoption of such security mechanisms.

## 4.2.1 Securing Applications in the Rich Execution Environment

### Related publication

📄 Vincent Giraud and Guillaume Bouffard. “Faulting original McEliece’s implementations is possible”, In: *SILM* 2023 [Article in PDF](#) [GB23]

As previously discussed, the **REE** remains an open and potentially untrusted environment, raising security questions when executing sensitive applications. These applications often involve cryptographic operations to ensure data security. Due to the complexity of developing a generic implementation that supports multiple platforms, cryptographic libraries are often embedded directly within the application. As a result, software cryptographic implementations deployed within the **REE** are highly exposed to attacks due to the lack of strong isolation mechanisms. Without access to a **TEE** or hardware security

modules, such as hardware cryptographic accelerators, developers must assume an adversary capable of fully controlling the execution environment, as in the white-box attack model [Cho+02].

In this context, the Giraud’s Ph.D. thesis [Gir24] investigates how a sensitive application can be executed as securely as possible within an uncontrolled execution environment. Our research primarily focuses on securing sensitive data handled by cryptographic services, particularly in the presence of a white-box attack model.

To protect sensitive assets within an application, it is essential to conceal its implementation. This involves applying code obfuscation techniques to make the program logic harder to analyze and modifying data representations to render sensitive information unintelligible to an attacker. These approaches increase the complexity of reverse engineering, significantly hindering an adversary’s ability to extract cryptographic secrets or manipulate critical operations.

Building on this, we investigate the use of an obfuscated application as a mean to store and manipulate sensitive assets within an uncontrolled execution environment. Our study explores how such an application can protect cryptographic keys and other critical data against extraction attempts, while ensuring their secure usage despite the absence of hardware-based security mechanisms. This contribution is labeled (C8) in Figure 4.1.

**White-box cryptographic implementations** are particularly exposed to software-based attacks due to the adversary’s full visibility and control over the execution environment, making them highly vulnerable to various forms of exploitation [Cho+02]. In this context, it is possible to transpose hardware attacks [Boc+19] onto software implementations, leveraging binary instrumentation techniques to extract secrets. Originally developed to break the security of SEs, hardware attack techniques can now be transposed to the binary level, enabling state-of-the-art methodologies to evaluate and reinforce the security of **white-box cryptographic implementations**.

To explore this threat, we applied fault injection attacks to the reference implementation of the McEliece cryptosystem on Arm-based platforms [Pet+15]. Our goal was to assess its resilience and, more broadly, to investigate how a McEliece-based **white-box cryptographic implementation** could be designed while accounting for the risk of fault injection transposition. This study highlights how attackers can exploit software-based fault injection to compromise cryptographic keys, emphasizing the need for countermeasures tailored to white-box security models.

Our findings demonstrate that even theoretically robust cryptographic algorithms, such as McEliece, can be vulnerable when executed in an untrusted REE. The attack presented in [GB23] exploits a fault injection-based approach to modify program instructions at runtime, leading to information leakage and a significant reduction in key entropy. Specifically, our attack reduced the entropy of the secret key by 40% to 70%, drastically narrowing the search space and making key recovery feasible. These results further reinforce the need for countermeasures that address both traditional fault injection techniques and software-based adversarial models.

To mitigate these threats, we propose a modified implementation of the McEliece cryptosystem designed to resist such attacks. Our approach leverages structural modifications in the decryption process, incorporating techniques to minimize the impact of fault injections. This contribution is part of broader efforts to enhance the security of sensitive software in the white-box attack model, where assumptions about the integrity of the OS no longer hold.

### 4.2.2 Synthesis

In this section, we examined the security challenges of executing sensitive applications within an untrusted REE, where adversaries have full control over the execution environment. Because application developers may lack access to a TEE or embedded hardware security mechanisms, they must rely on obfuscation techniques to protect sensitive assets against a wide range of attacks, particularly under white-box adversaries. This approach is especially relevant when third-party developers cannot offload *business security functions* to the TEE, as described in Figure 1.2, or when applications must operate securely within a REE without hardware-based protections.

To analyze the limitations of these obfuscation techniques, we investigated how hardware fault injection methods could be transposed to software implementations using binary instrumentation [GB23]. Our study focused on an obfuscated application used to store and manipulate sensitive assets in an uncontrolled execution environment. Through our analysis of cryptographic key storage and protection mechanisms, we demonstrated that even theoretically robust cryptographic algorithms remain vulnerable in such contexts. Specifically, our attack reduced the secret key's entropy by 40% to 70%, significantly narrowing the search space and making key recovery feasible via brute force. This extends previous research that had only managed to extract the public key [Cay22].

In light of these discoveries, we developed a revised McEliece cryptosystem implementation specifically engineered to resist fault injection attacks [GB23]. This solution is part of broader efforts to strengthen the protection of sensitive software in white-box environments, where the OS can no longer be assumed entirely trustworthy.

This topic remains crucial whenever developers cannot ensure that a sensitive application can offload its execution to a TEE nor verify the integrity of the REE. In such scenarios, relying on a *white-box cryptographic implementation* is one potential approach, but it must be combined with binary obfuscation techniques to protect the embedded secrets. Securing cryptographic assets under these conditions presents a significant challenge, demanding robust defenses capable of withstanding software-based and white-box adversaries.

## 4.3 Rich Execution Environment Hardware Security Challenges

### Associated contextual elements of this research field

- Collaborations:** Jessy Clédière (CEA/Leti, Grenoble, France), Ronan Lashermes (INRIA/LHS, Rennes, France), Maria Méndez Real (Université de Bretagne Sud, Lab-STICC, Lorient, France) and Jean-Christophe Prévotet (IETR & INSA, Rennes, France),
- Publications:** [Gon+25; Tro+21]
- Supervisions:** Thomas Troughkine [Tro21] and Gwenn Le Gonidec [Gon26]. Ph.D. theses

### Related publications

- 📖 Gwenn Le Gonidec, Guillaume Bouffard, Jean-Christophe Prévotet, and Maria Méndez Real. “Do Not Trust Power Management: A Survey on Internal Energy-based Attacks Circumventing Trusted Execution Environments Security Properties”, In: *ACM Transactions on Embedded Computing Systems* 2025 [Article in PDF](#) [Gon+25]
- 📖 Thomas Troughkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, and Guillaume Bouffard. “Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models”, In: *Journal of Cryptographic Engineering* 2021 [Article in PDF](#) [Tro+21]

The REE runs on an application processor that integrates various hardware security mechanisms designed to enhance software protection. These include the MMU for enforcing memory isolation, CFI to prevent control-flow hijacking, and additional features such as PAC or CHERI to mitigate memory corruption vulnerabilities. While these mechanisms strengthen software security against conventional exploits, they are not designed to withstand fault injection attacks.

In the Troughkine’s Ph.D. thesis [Tro21], we demonstrated that hardware attacks can significantly compromise these security mechanisms [Tro+21], potentially bypassing their protections or altering their expected behavior. Fault injection techniques can induce unintended modifications in the execution of critical security components [Kas23; Rot24], raising concerns about the reliability of these protections when deployed on end-user devices, where adversaries may have physical access. These works are described in Section 3.3.1.

Complementing this research, the Ph.D. thesis of Gonidec [Gon26], described in Section 3.4, investigates how integrated modules within an application SoC, such as PMUs, can be leveraged to mount hardware attacks against the TEE. Application SoCs embed the application CPU, which executes the REE, alongside numerous hardware modules that enhance performance and functionality. As illustrated in Figure 3.2, these include components such as GPU, VPU, and power management units, which interact with the execution environment. Specifically, this work examines the feasibility of remotely triggering fault injection attacks from PMUs to compromise sensitive applications [Gon+25], highlighting novel threats to the overall security of both TEE and REE environments.

Beyond their impact on specific hardware security mechanisms, fault injection attacks can also undermine the integrity of the REE itself. In a scenario where the REE is initially trusted, an attacker

could exploit fault injection techniques to corrupt its execution, either to gain unauthorized access to sensitive information [Fan+22] or to manipulate the system for espionage purposes. By inducing controlled faults, adversaries may extract confidential data processed within the REE or modify its security policies, enabling persistent monitoring and stealthy exploitation of the system. This issue is also explored in my ongoing research on fault injection attacks targeting TEEs, as discussed in Section 3.3.1 and presented in [ITB23].

Such attacks underscore the risks of relying solely on the REE for executing security-sensitive applications. The ability to compromise the REE through hardware-based manipulations reinforces the necessity of deploying additional layers of protection, such as leveraging a TEE or adopting fault-resistant software and hardware countermeasures.

## 4.4 Conclusion and Perspectives

In this chapter, we explored the security challenges associated with executing sensitive applications within the REE, an open and potentially untrusted environment. We highlighted the risks posed by software-based and hardware-assisted attacks, particularly in the context of cryptographic implementations deployed without access to a TEE or secure hardware modules.

Our study demonstrated that white-box cryptographic implementations are highly vulnerable to fault injection attacks, as adversaries can transpose hardware fault techniques to software through binary instrumentation. We specifically analyzed the McEliece cryptosystem, showing that even post-quantum algorithms can be compromised when executed in an unprotected REE. These findings emphasize the need for robust countermeasures, including obfuscation techniques, runtime protections, and fault-resistant cryptographic designs.

Beyond cryptographic applications, we also examined the security of hardware security mechanisms embedded in application processors, such as MMU, CFI, and PAC. We discussed how fault injection attacks can undermine these protections, ultimately compromising the integrity of the REE itself. The ability to corrupt the REE through such attacks raises critical concerns regarding data confidentiality, system integrity, and the feasibility of secure execution in untrusted environments.

These findings underscore the necessity of designing security architectures that surpass conventional assumptions regarding REE OS trustworthiness. While leveraging a TEE remains a preferred approach, it is not always feasible for developers who lack access to dedicated hardware protections. Consequently, enhancing software security mechanisms against both logical and hardware fault-based attacks continues to be a pressing concern. Future research should explore practical countermeasures that effectively mitigate these threats without compromising performance or usability.

Nevertheless, although the security of sensitive applications executed in the REE is not among the primary research perspectives outlined in Chapter 5. This topic remains critically important, particularly for third-party developers unable to verify the integrity of the REE or offload execution to a TEE, yet my main focus will be directed toward other aspects of hardware and software security.

## Chapter 5

# Conclusion and Perspectives

The work presented in this manuscript began in the [LSC](#) and continued in the [LAM](#) at the [ANSSI](#). In the Hardware Security Lab, I focused on analysing the security of components against hardware and software attacks. Later, in the Hardware and Software Architectures Lab, I extended this research to examine the security of components within their ecosystems, addressing the more complex architectures of modern devices.

At the end of 2014, when I started working at [ANSSI](#), [SEs](#) were the primary components used as [RoTs](#) to run security functions. By the late 2010s, a significant shift occurred in the field of security. Sensitive operations gradually began to be executed outside of [RoTs](#), directly within application [CPUs](#) in the [TEE](#) environment. This change was driven by the increasing complexity and performance demands of sensitive applications. As a result, new architectures were needed to extend trust beyond hardware [RoTs](#), leading to the adoption of systems based on a [CoT](#).

In my research activities, I have focused on the security of software implementations embedded within each element of the [CoT](#). This involves analyzing vulnerabilities and developing countermeasures to enhance the overall security of software, from the hardware [RoT](#) to the [REE](#). In this manuscript, I described my contributions to software security within both the hardware [RoT](#) and the [TEE](#), addressing threats from both hardware and software attacks.

Beyond the [TEE](#), I also explored the security challenges of executing sensitive applications within the [REE](#), where adversaries have full control over the execution environment. This research is particularly relevant when developers do not have access to a [TEE](#) or secure external components to protect their sensitive assets. In such scenarios, ensuring the integrity and confidentiality of sensitive operations requires alternative approaches, such as software-based protections and implementations resilient to hardware attacks. While this topic remains critical in certain contexts, it does not fall within the long-term research perspectives outlined in [Section 5.2](#).

To conclude this manuscript, [Section 5.1](#) provides a synthesis of the main research activities and

highlights the contributions made throughout the different elements of the CoT. This summary not only recalls the results presented in each chapter but also emphasizes their coherence within a broader research agenda. Section 5.2 outlines the future directions of my research. I will continue to enhance the security of the CoT by participating in initiatives aimed at improving the design security of hardware RoTs and TEEs based on open-source architectures. Additionally, I will extend my research to focus on designing architectures for safety-critical systems, which are becoming increasingly connected. These architectures now implement CoT [Glo23], similar to the one architecture introduced in Chapter 1. In safety-critical systems, vulnerabilities not only compromise data security but also pose significant risks to human life.

## 5.1 Summary of Activities Introduced in this Manuscript

This manuscript provides an overview of my research activities in the field of embedded software security, focusing on defending against both software and hardware attacks across different layers of the CoT. As introduced in Chapter 1, my work spans the security of hardware RoTs, which operate in constrained environments with minimal attack surfaces, the protection of TEEs within complex application CPUs, and the challenges associated with executing sensitive applications in the REE, where adversaries have full control over the execution environment. By analyzing vulnerabilities and developing countermeasures at each level of the CoT, this research contributes to strengthening the security of embedded systems in diverse execution contexts, from highly isolated secure elements to open and potentially untrusted environments.

Chapter 2 presents my research on the software security of hardware RoTs, focusing on SEs, which represent the most secure form of hardware RoT. Most SEs embed Java Card technology. During my Ph.D. thesis [Bou14], we extensively studied the implementations of JCVMs. Upon joining ANSSI, I expanded this work by examining the organizational assumptions underlying Common Criteria evaluations that are intended to guarantee the security of embedded JCVm implementations, specifically the reliance on a perfect BCv and a bug-free toolchain. In collaboration with the Thales and Serma ITSEFs, we investigated the BCv and toolchain, revealing previously undisclosed security vulnerabilities in these tools.

In addition to this, we explored the security of communication protocols, primarily focusing on the ISO/IEC 7816 stack, which is mostly deployed in SEs. This investigation did not reveal specific vulnerabilities; however, it highlighted the potential for implementation fingerprinting, which represents an area that requires further research.

Chapter 3 details my contributions to the security of TEEs, which are embedded within application CPUs. A TEE serves as a secure software environment that isolates sensitive and high-performance programs, ensuring their protection during execution. However, the security of a TEE also relies on the hardware security foundation.

In the mid-2010s, hardware attacks originally designed to target **SEs** were redirected towards application **CPUs** running **TEEs** [MBB16; TSW16; Vas+17]. These **CPUs** are not inherently designed to resist hardware attacks. According to the **protection profile** [Glo20b], **TEE** implementations must take hardware attacks into account.

When I began this research in 2017, the exploitability of fault injection attacks on application **CPUs** was a subject of ongoing discussion. To address this, we focused on understanding the possibilities and consequences of these attacks in **TEE** environments. We followed two approaches to achieve this goal. First, to protect **TEE** implementations running on **COTS components** without hardware security protections, we characterized fault effects at the **ISA** level. This work was part of the research conducted in **Trouchkine's** Ph.D. thesis [Tro21]. When the hardware implementation is known, we can characterize fault effects directly at the hardware level. This work was initiated during **Marotta's** Ph.D. thesis [Mar25], where we focused on understanding fault effects at the electronic logic level. These efforts will help to bridge the gap between software security and hardware vulnerability analysis, contributing to the development of more resilient **TEEs** capable of resisting fault injection attacks.

Furthermore, as part of **Gonidec's** Ph.D. thesis [Gon26], we are examining the security risks posed by **PMUs**, which have become sensitive components in modern **SoCs**. According to the **protection profile** for embedded **SEs** [Eur22], other modules within an **SoC** can also be a source of hardware attacks. **TEEs** running on application **CPUs** are therefore potential targets of these modules, which can be manipulated from a compromised **REE**. In the context of **Gonidec's** thesis, we focused specifically on **PMUs**, as they share capabilities similar to other fault injection mediums, such as modifying the frequency and voltage supply. **PMUs**, which control power management and are integrated alongside application **CPUs**, introduce new attack vectors, including fault injections and remote manipulations, that can compromise **TEEs**. Our research aims to characterize these vulnerabilities and propose high-level countermeasures to mitigate these emerging threats.

**Chapter 4** extends this analysis to the security challenges of the **REE**, where sensitive applications may need to operate in an environment fully controlled by adversaries [Cho+02]. Unlike **TEEs** or **SEs**, the **REE** lacks strong isolation mechanisms, making it vulnerable to a wide range of attacks, including software exploitation and hardware fault injections. This issue has been further investigated in **Giraud's** Ph.D. thesis [Gir24], which explores the security of cryptographic implementations in untrusted execution environments and evaluates their resilience against advanced attack techniques.

Our research explored how cryptographic implementations deployed in this context can be compromised and what countermeasures could be considered. We demonstrated that fault injection attacks can be transposed to software implementations using binary instrumentation techniques [GB23], leading to key recovery and system subversion. While this topic remains relevant in scenarios where a **TEE** or external security components are unavailable, it does not fall within the long-term research directions outlined in this manuscript.

This manuscript presents a comprehensive analysis of the security challenges faced by **CoTs**, covering

the software security of hardware **RoTs**, the vulnerabilities in application **CPUs** running **TEEs**, and the risks associated with executing sensitive applications within the **REE**. By addressing both high-level architectural defenses and low-level hardware vulnerabilities, my research contributes to strengthen the security of **CoTs** in high-performance devices, considering the global view from isolated secure elements to open execution environments.

## 5.2 Perspectives

Building on the research activities presented in this manuscript, I will focus on two main research directions over the coming years to enhance the security of the **CoT** and its implementation in the safety-critical systems.

### 5.2.1 Towards a Secure and Reliable Chain of Trust

My research has primarily focused on the security of embedded software, addressing both software and hardware attacks. The security of software also fundamentally depends on the security of the hardware it runs on. Even the most secure software can be compromised by vulnerabilities in hardware components [Bit+21; Law10; Raz11; Rou+19; Slu+23]. To improve the security of runtime environment, I aim to study software security through the scope of the security provided by the underlying hardware execution environment.

To establish trust in each element of the **CoT**, most of my research activities have focused on analyzing the security of software implementations on **COTS components**, where the underlying hardware designs are not disclosed and remain unknown. Gaining partial insights into these architectures typically requires formal agreements with several entities, which limits the ability to develop sovereign solutions. With the advent of the RISC-V **ISA**, which is royalty-free and under open-source licence, several open-source implementations of hardware **RoTs** [Low24; Ope24b], application processors [NLn24; Ope24a], and **TEEs** [Lee+20; Sch+23b] layer are now available, providing valuable insights into such architectures. However, these solutions often have incomplete security features and may not have been evaluated under security schemes.

My research perspectives will focus on developing a secure and reliable **CoT** hardware architecture. Thanks to the RISC-V **ISA**, it is now possible to design each element of the **CoT** as a set of blocks from open-source repositories and proprietary implementations. Viewing secure hardware architecture as a combination of blocks developed by different sources represents an original and challenging approach. Open-source blocks benefit from community contributions and often embed cutting-edge functionalities. However, their quality may be lower than that of proprietary blocks, as they may not fully comply with industry standards or have the same level of maturity. The use of open-source blocks must be carefully analyzed from both functional and security perspectives to ensure their suitability within a secure **CoT**.

As introduced in [Chapter 1](#), the **REE** is a layer where high-performance and *untrusted applications* run. These untrusted applications generally come from different developers, and typically, only the application binary is provided. Due to its large attack surface, the **REE** layer is extremely difficult to secure. Some application **CPUs** embed hardware mechanisms to protect software against classic development vulnerabilities, such as control-flow or memory access violations. For example, to prevent software bugs related to memory management from being exploited, certain mechanisms are embedded at the hardware runtime level to provide tagged memory pointers [[Zel+08](#)]. Recently, **CHERI** [[Woo+14](#)] has emerged with significant support from component designers and software developers. This approach primarily protects against attacks exploiting memory access bugs, such as buffer or heap overflows and underflows.

While protecting memory accesses and enforcing control-flow integrity are essential first steps, other vulnerabilities, such as data parsing errors, logical flaws in program architecture, or misunderstandings of implemented specifications, remain beyond the scope of these mechanisms and require additional measures. Furthermore, approaches to protecting applications in the **REE** via hardware mechanisms often require modifying and recompiling a significant part of the software execution environment. Enforcing the use of these hardware blocks for sensitive applications is particularly challenging if the entire environment is not designed to support them. This process becomes even more complex when certain software parts, such as drivers, are no longer maintained, making it difficult to adapt existing software stacks to different architectures.

To protect sensitive high-performance applications, a more effective solution is to embed them into the **TEE** layer, which is designed to execute only *trusted applications*. Software embedded in the **TEE** layer is considered trusted if it undergoes a validation process. This process can take two forms: either a strict security audit, such as a security certification or a rigorous code review, or a combined logistical and technical validation. In the latter case, a trusted third party approves the application to be loaded after several static checks, while a runtime environment ensures that the application remains segregated. This runtime should be evaluated under specific **protection profiles**, capable of supporting the ability to load applications from different developers. This approach is similar to the one deployed in the security model of Java Card [[Ora21c](#)]. This contrasts with the *untrusted applications* that typically run in the **REE** layer.

Current **REE** and **TEE** software layers share the same hardware runtime environment. The **REE** contains a vast amount of existing code, including widely used **OSes** like Android, iOS, Linux, and Windows, along with numerous applications. Since the **TEE** layer runs on the same architecture as the **REE** layer, it also benefits from a wealth of existing code, including libraries and development tools such as compilers, which eases development and integration. These software stacks are designed for widely deployed architectures, and adapting them to new architectures is an extremely costly operation.

Troughkine's Ph.D. thesis [[Tro21](#)] demonstrates that the hardware layer designed for the **REE** environment remains a point of vulnerability against fault injection attacks. Since the **TEE** relies on the same hardware as the **REE**, any features introduced to the applicative **CPU** aimed at improving the performance of the **REE** can directly impact the security of the **TEE**, potentially creating new exploitable

hardware vulnerabilities. Contrary to the requirements defined in the **TEE protection profile** [Glo20b], most current implementations do not address the risks of hardware attacks. Sharing hardware between security-oriented (**TEE**) and non-security-oriented (**REE**) layers opens vulnerabilities, as we have shown in [Gon+25].

I believe that achieving a high-performance hardware runtime capable of mitigating hardware attacks is impossible without modifying the hardware architecture on which the **REE** environment runs. To enhance security, the **CoT** architecture must therefore be reconsidered.

To improve its security, the **TEE** software layer should run on dedicated hardware that tends towards the same security level as the hardware **RoT**, while supporting high-performance applications and maintaining a low energy footprint. This new type of hardware block would be embedded within the application **SoC**, positioned as close as possible to the main **CPU** running the **REE**. To ensure ease of development and maintenance, this architecture should be supported by unmodified mainstream tools to facilitate its integration into existing software ecosystems.

The secure sub-system in **SoC protection profile** [Eur22] describes the security requirements for embedding a hardware **RoT** within an applicative **SoC**, where some micro-architecture blocks are shared between secure and non-secure blocks. I believe this **protection profile** could be applied to dedicated micro-architecture blocks specifically designed for secure, high-performance, and energy-efficient purposes, as required by the **TEE** layer.

Building upon these observations, this work aims to extend the activities carried out within the scope of **research question 2**, described in **Section 1.3**. In **Chapter 3**, existing implementations have been studied from the perspective of fault injection attacks. From these insights, my research perspectives will focus on designing a specific high-performance, energy-efficient, and secure hardware layer capable of running a **TEE** environment while embedding protections against hardware attacks. Integrating such protections within an environment hosting the **TEE** seeks to provide a secure foundation for sensitive, high-performance applications with low energy consumption. To achieve this, my future research directions will be split into two main parts: first, analyzing the countermeasures embedded in hardware **RoTs** against hardware attacks, based on hardware and software blocks from different developers, as outlined in **Section 5.2.1.1**; and second, securely adapting these mechanisms to meet the constraints of high-performance and energy-efficient **TEE** environments, as detailed in **Section 5.2.1.2**.

### 5.2.1.1 Understanding How to Secure Hardware Root of Trust ...

#### Associated contextual elements of this research field

**Collaborations:** Luc Bonnafox (ANSSI, Paris, France), Damien Couroussé (CEA/List, Grenoble, France), Mathieu Jan (CEA/List, Saclay, France), and Philippe Trébuchet (ANSSI, Paris, France)

**Supervisions:** Angie-Sofia Bikou's internship [Bik24] and Jonah Alle Monne's Ph.D. these [All27].

The emergence of open-source hardware **RoTs** provides a unique opportunity to analyze and enhance their security. Existing implementations, such as OpenTitan [Low24] and CV32E40S [Ope24b], can be embedded or adapted to be used as hardware **RoTs** [Sch+23a]. These designs offer valuable insights into their architecture and their embedded countermeasures. However, to serve as secure and reliable hardware **RoTs**, they should have a high-security level. Many of these solutions lack industrial maturity and have not been evaluated under security schemes.

My upcoming research will focus on gaining a comprehensive understanding of current hardware **RoTs**, examining their design, security features, and potential weaknesses. This analysis, combined with the specific requirements for state-use products, will serve to identify best practices and areas for improvement. Once the study of hardware **RoT** implementations is complete, I aim to investigate how to integrate a hardware **RoT** into an applicative **SoC**, following the guidelines outlined in the *secure sub-system in SoC protection profile* [Eur22]. This **protection profile** redefines the concept of a hardware **RoT** from being a standalone secure enclave containing isolated hardware modules for sensitive operations to a design integrated within an applicative **SoC**, where some hardware modules may be shared with non-secure elements. Currently, only two products [Arm22; SAM24] have been successfully evaluated under the Common Criteria scheme. The limited number of evaluated products highlights the complexity of implementing this **protection profile** and the challenges in guaranteeing security properties when transitioning from a traditional standalone hardware **RoT** to an integrated system within an applicative **SoC**.

To achieve a hardware **RoT** that is both secure and reliable, it is essential to analyze the security of existing open-source projects to understand their limitations and propose enhancements. Part of this research is conducted within the **PePR Arsene** funding project, where I contribute to the work package 1, focusing on secure RISC-V **CPU**.

**OpenTitan Security Analysis** First, a part of this research will focus on the OpenTitan project, an open-source hardware **RoT** designed as an implementation of a **SE**, with fewer features than mature proprietary implementations and no security evaluation. Our study of OpenTitan's security is structured around two main axes. The first axis focuses on the analysis of embedded software security, aiming to evaluate and harden the software against practical, real-world requirements. The second axis involves analyzing the security of the hardware execution environment, assessing the resilience of the hardware layer when paired with a realistic software environment.

Through these efforts, we seek to both deepen our understanding of and contribute to the security of OpenTitan's implementations. Moreover, this work aims to challenge and refine **ANSSI**'s security recommendations, thereby improving the security standards applied to evaluated products. This work will involve internal projects within **ANSSI**, including contributions from internships such as those led by Bikou [Bik24], and will continue through future internships and collaborations.

**CV32E40S Security Analysis** Next, another part of this research will be the analysis of the security of CV32E40S, an open-source **CPU** which embeds countermeasures against hardware attacks. Unlike the **CPU** embedded in the OpenTitan project [Low24], CV32E40S is close to those embedded in proprietary

SE implementations. This open-source project may serve as main CPU for various hardware RoTs and offers an opportunity to analyze and enhance its security mechanisms.

My research will particularly build on insights from a research partnership between ANSSI and CEA where the Alle Monne [All27]’s Ph.D. thesis is conducted. The goal of this joint work is to understand the implementation of these countermeasures, their cover and propose enhancements to reinforce the mechanisms against hardware attacks embedded in hardware RoT.

For this purpose, the embedded countermeasures are analyzed using  $\mu$ ArchiFI [Tol+23], a tool based on formal approaches operating at the RTL or netlist level. Developed during Tollec’s Ph.D. thesis [Tol24], this tool aims to identify potential paths that could compromise declared security properties, described using formal specifications, with respect to fault models.

**Analysis of Security Properties at Every Stage of the Design Process** Finally, this research area will extend to analyzing the security properties of hardware RoTs from the RTL level down to the ASIC mask level. This comprehensive approach aims to provide a detailed understanding of the hardware’s security features throughout its lifecycle.

In parallel, in collaboration with CEA/List and CEA/Leti, we have begun analyzing the gap between fault attacks simulated at the RTL or netlist levels and those observed experimentally at the silicon (ASIC) level. For this analysis, we use the VASCO<sup>1</sup> proof-of-concept hardware RoT. Characterizing these differences leverages the methodology developed during the thesis of Trouchkine [Tro21], which measures the effects of faults starting from the instruction-set level. This approach directly connects observable architectural faults to RTL-level simulations, a connection made possible by using an ASIC environment where all components are fully controlled.

Building upon this experimental observation, the simulation-based approach proposed by Alle Monne [All27] uses formal methods to anticipate fault effects before they appear at the silicon level. By combining *experimental observations* and *simulation*, we aim to propose a robust and generic methodology capable of enhancing the reliability of security assessments and optimizing hardware countermeasure design.

Additionally, the thesis of Marotta [Mar25] investigated fault injection attacks at the logic gate level, providing valuable insights into the transposition of security mechanisms from the RTL level down to netlist and ASIC. Leveraging this work, we plan to study how hardware countermeasures can be consistently maintained throughout the hardware synthesis process. The goal is to guarantee a constant security level despite successive transformations involved in hardware synthesis, by identifying necessary adjustments to prevent degradation of the implemented protections.

Ultimately, this research seeks to provide methodologies applicable to environments hosting high-performance, energy-efficient TEEs while maintaining robust security assurances.

---

1. VASCO is an ASIC prototype developed by CEA to evaluate innovative security components.

### 5.2.1.2 ... To Enhance Trusted Execution Environment Security to Mitigate Hardware Attacks

#### Associated contextual elements of this research field

<b>Collaborations:</b>	Mariá Mendez Real (Lab-STICC, Lorient, France), Jean-Christophe Prévotet (IETR/INSA, Rennes, France)
<b>Publications:</b>	[Gon+25]
<b>Supervisions:</b>	Gwenn Le Gonidec's Ph.D. these [Gon26].

According to the [protection profile \[Glo20b\]](#), the [TEE](#) layer must address the threat of hardware attacks. However, as the [TEE](#) and [REE](#) share the same performance-oriented [CPU](#), which lacks dedicated security protections, mitigating such threats is particularly challenging. Designing secure applications for a [TEE](#) that considers vulnerabilities from the hardware layer requires specific risk assessments and advanced security measures, demanding expertise from developers in both application and hardware security. Developing a secure application is a highly complex task, and finding developers with expertise in software security, application-specific security, and hardware security is exceptionally rare. Therefore, relying solely on the developer's ability to ensure security is not a viable solution. To ensure the security of the [TEE](#) against hardware attacks, I believe it is essential for the [TEE](#) to have a dedicated [CPU](#) that is both secure and high-performance, as outlined in the *secure sub-system in SoC protection profile* [Eur22].

[CPUs](#) running a [TEE](#) are integrated into modern [SoCs](#). Those [SoCs](#) embed additional hardware modules, such as power management systems or modems, which increase the system's complexity compared to hardware [RoT](#) implementations like [SEs](#). This added complexity opens new pathways for exploiting attack vectors [Ayo+24; Cam+18; TSS17; Zha+18].

To secure [TEE](#) implementations, it is essential to understand the impact of hardware attacks originating from or targeting these additional hardware modules. Developing such an understanding the requirements to better protect [TEE](#) implementations against these threats. This work has begun as part of [Gonidec's](#) Ph.D. thesis [Gon26], where we study power management mechanisms that could potentially leak information or enable fault attacks. Additionally, within the [PTCC FORWARD](#) funding project, we are exploring hardware- and software-level countermeasures to enhance the protection of the [TEE](#) against hardware attacks.

The *secure sub-system in SoC protection profile* [Eur22] primarily outlines how to secure a set of hardware blocks that constitute a hardware [RoT](#) embedded within an applicative [SoC](#). Building on the insights gained from designing a secure hardware [RoT](#), introduced in the [Section 5.2.1.1](#), I believe that similar principles can be applied to develop a dedicated hardware layer for [TEEs](#), combining security with high-performance. This ongoing work extends research conducted during [Gonidec's](#) Ph.D. thesis [Gon26], the [PTCC FORWARD](#) project, and additional internal projects.

## 5.2.2 Safety-Critical Systems need High-Level of Cybersecurity

### Associated contextual elements of this research field

**Collaboration:** Philippe Trébuchet (ANSSI, Paris, France)

**Publication:** [TB25]

*Safety-Critical Systems* are those whose failure could result in significant harm to people, property, or the environment. They are employed in multiple sectors (e.g., medical, industrial, transportation) where human lives and critical infrastructures are directly at stake. National and international laws, along with industry standards, mandate robust mechanisms to enhance the safety and security of these systems. Such mechanisms aim to prevent failures, detect anomalies, and ensure correct responses to avoid catastrophic outcomes. As these systems become increasingly connected to external environments, the scope and complexity of their security requirements grow correspondingly.

This work introduces a new research question: *How can we effectively bridge security and safety?* This question extends the scope of the three research questions presented in [Section 1.3](#), broadening my investigation to *critical systems* requiring high levels of both security and safety. By exploring this challenge, my goal is to propose solutions that simultaneously ensure the protection, defense, and resilience of critical architectures, while strictly complying with the rigorous safety constraints inherent to these environments.

### 5.2.2.1 Criticality of Safety and Cybersecurity in Connected Environments

Modern vehicles, often referred to as *connected vehicles*, illustrate well the blend of stringent safety requirements (to protect lives) and the imperative for robust cybersecurity measures. These systems rely on numerous connectivity technologies (Bluetooth, Wi-Fi, cellular networks, etc.) and sophisticated sensing capabilities (cameras, LiDAR, radar, and ultrasonic sensors). While they offer enhanced comfort, functionality, and safety features such as [Advanced Driver Assistance Systems \(ADAS\)](#) and autonomous driving, they also bring new cybersecurity risks. Indeed, the increase in interconnectivity and data exchange broadens the attack surface, exposing the system to remote and physical threats alike [[Evd24](#); [Lab24](#); [MV13](#)].

For sensitive operations, connected vehicles cannot have their features disabled due to compliance with national and international regulations. These vehicles must maintain high standards of confidentiality, integrity, and availability. [ANSSI](#) thus has a critical role to play in assessing cybersecurity levels and promoting improvements aligned with security objectives.

### 5.2.2.2 Challenges in Safety-Critical Domains

Beyond the automotive domain, the increasing connectivity of safety-critical systems in medical and industrial settings also exposes them to potentially severe attacks. Medical devices such as pacemakers [[Hal+08](#)] and industrial control systems in critical infrastructures [[AZ23](#); [Kha+16](#); [Lan11b](#)] face

comparable threats. An attack on a pacemaker could endanger a patient's life, while a breach in an industrial control system could disrupt essential services.

In transportation, the high stakes of passenger and operational safety make the integration of cybersecurity and safety measures especially vital. Connected and autonomous vehicles, in particular, gather and process large volumes of sensor data and interacting with external services and other vehicles on the road. The presence of multiple communication channels, firmware updates, and data sharing in real time increases the risk of malicious intrusions that could undermine safety-critical features [Evd24; Lab24].

### 5.2.2.3 Chain of Trust in Safe-critical Environment

A connected vehicle can be viewed as an embedded system requiring a high guarantee of functional safety. Security in connected vehicles thus entails combining functional safety and cybersecurity while adhering to standards that are regularly updated. To address the complexity of continuously adding new features, GlobalPlatform [Glo23] proposes implementing a CoT: from a hardware RoT to a TEE, and then to a REE running a complex operating system such as Android Auto. This architecture aims at preserving the integrity of the execution environment while allowing performance-intensive yet sensitive tasks to run within the TEE.

Unlike hardware RoTs in smartphones or computers, those embedded in vehicles are often powered continuously, even when the vehicle is idle. This ensures that security services are always available to verify integrity across various operational states. However, permanent power also makes the vehicle an attractive target for attacks, including charging-port exploits on electric vehicles [Dud19] or physical compromise by an attacker with direct access [Mel24; OFI20; Wer+23]. Such threats can mirror attacks observed against SE [Wer+23], where an adversary might extract cryptographic keys or introduce malicious firmware, ultimately jeopardizing the vehicle's overall security. Incorporating protections against these attacks is essential for safeguarding of both cybersecurity and safety properties.

### 5.2.2.4 Case Study: Bluetooth Stack Analysis in a Connected Vehicle

#### Related publication

☒ Philippe Trébuchet and Guillaume Bouffard. “300 secondes chrono: prise de contrôle d'un infodivertissement automobile à distance”, In: *SSTIC 2025* [Article in PDF](#) [TB25]

As part of an internal ANSSI project, we analyzed a representative vehicle [TB25] used by both the French administration and private individuals. Among its wireless protocols, Bluetooth was identified as a critical point of vulnerability because it remains active even when no user is connected. We discovered a flaw that allows a remote, unauthenticated attacker to execute arbitrary code without occupant interaction. Such an exploit potentially opens the door to sending unauthorized CAN messages or performing other malicious actions once the REE is compromised<sup>2</sup>.

2. The RoT in this vehicle filters sensitive CAN commands, preventing certain attacks on safety-critical features [TB25].


In this particular vehicle, the hardware RoT is present but does not provide strong segregation from the REE. Some sensitive CAN messages are filtered by the RoT, preventing direct harm to safety-critical functions. Nevertheless, an attacker controlling the REE could still perform significant malicious actions. This underscores the importance of implementing a more complete CoT architecture [Glo23], where the hardware RoT and TEE form a robust boundary against potential compromises of the REE.

#### 5.2.2.5 Towards a Unified Perspective on Safety and Security

The exploration of security in connected vehicles leverages expertise from the analysis of hardware RoTs and TEEs, including knowledge of how software implementations fare against hardware-based attacks. As I aim to deepen my understanding of connected and autonomous vehicles, this research naturally extends to other critical sectors, medical and industrial, where similar risks emerge from the intersection of safety and security.

The broader objective is to examine how security requirements can coexist with the stringent demands of functional safety in these environments. Security introduces new design constraints, while safety mandates that these additions do not erode established reliability standards. By studying potential hardware and software attacks together, we can propose protection mechanisms that preserve both security and safety. This approach is especially crucial for next-generation electrified vehicles, where continuous communication with charging stations or roads further multiplies potential entry points for attackers.

#### 5.2.2.6 Toward more Secure Vehicules

Continuing my current research, I intend to deepen the analysis of hardware RoTs and TEEs in connected and autonomous vehicles, including scenarios involving fault injection attacks. Another focus will be on investigating how emerging infrastructures, such as in-motion charging, further expand the attack surface for electrified vehicles. This work is represented by the thesis highlighted in red , illustrated at the bottom of Figure 1.3, and focusing on mixing security and safety-critical CoT. It is entitled “*Security in a safety-critical environment*”.

In parallel, I plan to extend these studies to other domains, particularly medical and industrial sectors, where security and safety must coexist to protect both human lives and critical infrastructures. A key aspect of this work will be the advocacy for rigorous security evaluations of hardware RoTs according to recognized certification schemes, ensuring that the mechanisms protecting CoTs architectures remain robust against sophisticated threats. Addressing safety and security jointly remains a complex challenge, as their requirements may at times diverge or even conflict, requiring careful trade-offs to guarantee both reliable operation and strong protection.

Ultimately, my goal is to propose solutions that harmonize security with functional safety, ensuring that as systems become more interconnected, they also become more resilient against attacks. This balance is paramount in domains where the slightest compromise in safety or security can have immediate and severe repercussions.

## 5.3 Conclusion

To continue the work presented in this manuscript, I will explore approaches to enhance the security of **TEEs** against hardware attacks by leveraging previous research carried out on securing **SEs**. Initially designed as isolated secure vaults, **SEs** are primarily susceptible to external attacks. However, with the growing adoption of **TEEs** in application **SoCs**, hardware attacks have emerged as increasingly significant and complex threats. Indeed, such attacks may originate externally, using physical means outside the target application **SoC**, such as fault injection or side-channel analysis techniques, but also internally, by compromising hardware blocks embedded within the application **SoC** itself to target other system components. Understanding how to effectively protect these implementations is thus essential to ensure the robustness of trusted architectures.

My ongoing research on analyzing open-source hardware **RoTs** implementations, such as Open-Titan [Low24] and CV32E40S [Ope24b], aims to better understand the coverage of integrated security mechanisms and identify potential improvements. Ultimately, this analysis will help to define strategies for integrating these protections into high-performance, energy-efficient application processors running **TEEs** that are resilient against hardware attacks.

In parallel with this work on securing **TEEs**, I have initiated research on protecting critical systems, where combining security and safety is crucial, particularly in the context of connected vehicles. These systems, originally designed with a safety-first approach, must now incorporate security due to the increase in communication interfaces and the expansion of the attack surface. Research conducted on the security of communication protocols [TB25], notably through the analysis of vulnerabilities in the **REE** of connected and autonomous vehicles, highlights the necessity of strong separation between critical and non-critical components [Glo23].

Building upon the research conducted on hardware **RoTs** and **TEEs**, this work will adopt a holistic approach, addressing the protection of embedded software against both software and hardware attacks, while ensuring a balance between security and safety. The objective is to define architectures integrating **CoTs** tailored to safety-critical environments, where resilience and reliability are paramount.

This long-term research aims to shape the next generation of trusted architectures for safety-critical systems. By integrating **CoTs** capable of balancing performance, low energy consumption, and resistance to hardware attacks, this research addresses the growing challenges of connectivity and cybersecurity in embedded environments. The goal is to develop robust and modular solutions adapted to safety and security constraints, ensuring the reliability and resilience of tomorrow's safety-critical systems.



# Bibliography

- [Adv23] Advanced Micro Devices (AMD). *SEV Secure Nested Paging Firmware ABI Specification*. 56860. Version 1.55. Sept. 2023. URL: <https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf> (visited on 07/07/2025).
- [Ago+10] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. “When Clocks Fail: On Critical Paths and Clock Faults”. In: *Proceedings of the 9th International Conference of Smart Card Research and Advanced Application (CARDIS)*. Ed. by Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi- Cartigny. Vol. 6035. Lecture Notes in Computer Science. Passau, Germany: Springer, Apr. 2010, pp. 182–193. doi: [10.1007/978-3-642-12510-2\\_13](https://doi.org/10.1007/978-3-642-12510-2_13).
- [Alb21] Hans-Gerd Albertsen. *Certification Report. H1D3 Secure Microcontroller with Crypto Library v0.1.4*. Version 2.4. Nov. 2021. URL: <https://www.commoncriteriaportal.org/files/epfiles/%5BSTL%5D%5B2.4%5D%20Titan%20H1D3%20Security%20Target%20Lite%20v2.4.pdf> (visited on 07/07/2025).
- [AVR14] Vincent Alimi, Sylvain Vernois, and Christophe Rosenberger. “Analysis of embedded applications by evolutionary fuzzing”. In: *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*. Bologna, Italy: IEEE Computer Society, July 2014, pp. 551–557. ISBN: 978-1-4799-5312-7. doi: [10.1109/HPCSim.2014.6903734](https://doi.org/10.1109/HPCSim.2014.6903734).
- [All27] Jonah Alle Monne. “Formalization and Analysis of Countermeasures Against Fault Injection Attacks on Open-Source Processors”. PhD thesis. Grenoble, France: Université Grenoble Alpes, 2027. In preparation.
- [AZ23] Haya Altaieb and Rajnai Zoltan. “Malware Attacks on SCADA Systems: Assessing Risks and Strengthening Cybersecurity Measures”. In: *Proceedings of the 21st IEEE Jubilee International Symposium on Intelligent Systems and Informatics, (SISY)*. Pula, Croatia: IEEE, Sept. 2023, pp. 625–630. doi: [10.1109/SISY60376.2023.10417951](https://doi.org/10.1109/SISY60376.2023.10417951).
- [AF04] Tiago Alves and Don Felton. “TrustZone: Integrated Hardware and Software Security”. In: *Technology in-depth* 3.4 (2004).
- [AMD23] AMD. *AMD Ryzen™ Pro 7000 Series Mobile Processors*. Oct. 2023. URL: <https://www.amd.com/content/dam/amd/en/documents/products/processors/ryzen/7000/ryzen-pro-7000-security-whitepaper.pdf> (visited on 07/07/2025).
- [ANS23] ANSSI. *First Level Security Certification (Certification de Sécurité de Premier Niveau (CSPN)). HP Sure Start Hardware Root of Trust NPCX998HB0BX*. ANSSI-CSPN-2023/08. July 2023. URL: <https://cyber.gouv.fr/produits-certifies/hp-sure-start-hardware-root-trust-npcx998hb0bx-hpsshwnb21b0-0> (visited on 07/07/2025).
- [App16] Apple. *FairPlay Streaming*. Sept. 2016. URL: <https://developer.apple.com/streaming/fps/> (visited on 07/07/2025).

- [App24] Apple. *Apple Platform Security*. May 2024. URL: <https://support.apple.com/guide/security/welcome/web> (visited on 07/07/2025).
- [Arm12] Arm. *Jazelle direct bytecode execution support*. July 2012. URL: <https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/Application-Level-Programmers--Model/Jazelle-direct-bytecode-execution-support> (visited on 07/07/2025).
- [Arm17] Arm. *TrustZone technology for the ARMv8-M architecture Version 2.0*. Mar. 2017. URL: <https://developer.arm.com/documentation/100690/0200/ARM-TrustZone-technology> (visited on 07/07/2025).
- [Arm22] Arm. *Arm® CryptotIsland™-300P. Integrated Secure Element Security Target Lite*. NSCIB-CC-0397801-CR. Version 2.0. Aug. 2022. URL: [https://www.commoncriteriaportal.org/files/epfiles/arm\\_cryptoisland300p\\_integrated\\_secure\\_element\\_security\\_target\\_lite\\_107611\\_0000\\_02\\_en.pdf](https://www.commoncriteriaportal.org/files/epfiles/arm_cryptoisland300p_integrated_secure_element_security_target_lite_107611_0000_02_en.pdf) (visited on 07/07/2025).
- [Arr+20] Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. “Cryptographic Fault Diagnosis using VerFI”. In: *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. San Jose, CA, USA: IEEE, Dec. 2020, pp. 229–240. doi: [10.1109/HOST45689.2020.9300264](https://doi.org/10.1109/HOST45689.2020.9300264).
- [Ars+20] Muhammad Arsath, Vinod Ganesan, Rahul Bodduna, and Chester Rebeiro. “PARAM: A Microprocessor Hardened for Power Side-Channel Attack Resistance”. In: *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. San Jose, CA, USA: IEEE, Dec. 2020, pp. 23–34. doi: [10.1109/HOST45689.2020.9300263](https://doi.org/10.1109/HOST45689.2020.9300263).
- [AG21] Ever Atilano and Arnaud de Grandmaison. “Assessing the effectiveness of MCUBoot protections against fault injection attacks”. In: *Linaro Connect Virtual Connect Fall* (Sept. 2021). URL: <https://resources.linaro.org/en/resource/ibFLwRzhpZjBfvY5jhPypJ> (visited on 07/07/2025).
- [AF18] Gildas Avoine and Loïc Ferreira. “Attacking GlobalPlatform SCP02-compliant Smart Cards Using a Padding Oracle Attack”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.2 (May 2018), pp. 149–170. doi: [10.13154/tches.v2018.i2.149-170](https://doi.org/10.13154/tches.v2018.i2.149-170).
- [Ayo+24] Pierre Ayoub, Romain Cayre, Aurélien Francillon, and Clémentine Maurice. “BlueScream: Screaming Channels on Bluetooth Low Energy”. In: *Proceedings of the 40th Annual Computer Security Applications Conference (ACSAC '24)*. Waikiki, Honolulu, Hawaii, United States, Dec. 2024.
- [Bal+15] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. “DPA, Bitslicing and Masking at 1 GHz”. In: *Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Sept. 2015, pp. 599–619. doi: [10.1007/978-3-662-48324-4\\_30](https://doi.org/10.1007/978-3-662-48324-4_30).
- [Bar+06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. “The Sorcerer’s Apprentice Guide to Fault Attacks”. In: *Proceedings of the IEEE* 94.2 (Feb. 2006), pp. 370–382. doi: [10.1109/JPROC.2005.862424](https://doi.org/10.1109/JPROC.2005.862424).
- [Bar+22] Guillaume Barbu, Ward Beullens, Emmanuelle Dottax, Christophe Giraud, Agathe Houzelot, Chaoyun Li, Mohammad Mahzoun, Adrián Ranea, and Jianrui Xie. “ECDSA White-Box Implementations: Attacks and Designs from CHES 2021 Challenge”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.4 (Aug. 2022), pp. 527–552. doi: [10.46586/TCHES.V2022.I4.527-552](https://doi.org/10.46586/TCHES.V2022.I4.527-552).

- [BB113] Guillaume Barbu, Guillaume Bouffard, and Julien Iguchy-Cartigny. “La Sécurité Logique”. In: *Les Cartes à puce*. Ed. by Samia Bouzefrane and Pierre Paradinas. Hermes Science, 2013. Chap. 6, pp. 171–201. ISBN: 9782746239135.
- [Bar+11] Matthieu Barraud, Guillaume Bouffard, Nassima Kamel, and Jean-Louis Lanet. “Fuzzing on the HTTP protocol implementation in mobile embedded web server”. In: *C&ESAR*. Rennes, France, Nov. 2011, pp. 14–27.
- [BB23] Lejla Batina and Ileana Buhan. “Side-Channel Attacks”. In: *Encyclopedia of Cryptography, Security and Privacy*. Ed. by Sushil Jajodia, Pierangela Samarati, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, June 2023, pp. 1–4. ISBN: 978-3-642-27739-9. DOI: [10.1007/978-3-642-27739-9\\_1795-1](https://doi.org/10.1007/978-3-642-27739-9_1795-1).
- [Bel20] Yanis Belkheyar. “Authenticated Disk Encryption”. Master’s thesis. Paris, France: Université Paris 7, Sept. 2020.
- [Ber+14] Reinhard Berlach, Michael Lackner, Christian Steger, Johannes Loinig, and Ernst Haselsteiner. “Memory-efficient on-card byte code verification for Java cards”. In: *Proceedings of the 1rd Workshop on Cryptography and Security in Computing Systems (CS2@HiPEAC)*. Ed. by Jens Knoop, Valentina Salapura, Israel Koren, and Gerardo Pelosi. Vienna, Austria: ACM, Jan. 2014, pp. 37–40. DOI: [10.1145/2556315.2556323](https://doi.org/10.1145/2556315.2556323).
- [BRS17] Swarup Bhunia, Sandip Ray, and Susmita Sur-Kolay, eds. *Fundamentals of IP and SoC Security. Design, Verification, and Debug*. 1st ed. Springer Cham, Feb. 2017. ISBN: 978-3319934631. DOI: [10.1007/978-3-319-50057-7](https://doi.org/10.1007/978-3-319-50057-7).
- [Bik24] Angie-Sofia Bikou. “Analysis of an Open-Source Secure Component Architecture”. Master’s thesis. Paris, France: Sorbonne Université, Sept. 2024.
- [Bin+08] Walter Binder, Martin Schoeberl, Philippe Moret, and Alex Villazón. “Cross-Profiling for Embedded Java Processors”. In: *Proceedings of the 5th International Conference on the Quantitative Evaluation of Systems (QEST)*. Saint-Malo, France: IEEE Computer Society, Sept. 2008, pp. 287–296. DOI: [10.1109/QEST.2008.39](https://doi.org/10.1109/QEST.2008.39).
- [Bit+21] Otto Bittner, Thilo Krachenfels, Andreas Galauner, and Jean-Pierre Seifert. “The Forgotten Threat of Voltage Glitching: A Case Study on Nvidia Tegra X2 SoCs”. In: *Proceedings of the 18th Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. Milan, Italy: IEEE, Sept. 2021, pp. 86–97. DOI: [10.1109/FDTC53659.2021.00021](https://doi.org/10.1109/FDTC53659.2021.00021).
- [Boc+19] Estuardo Alpirez Bock, Joppe W. Bos, Chris Brzuska, Charles Hubain, Wil Michiels, Cristofaro Mune, Eloi Sanfelix Gonzalez, Philippe Teuwen, and Alexander Treff. “White-Box Cryptography: Don’t Forget About Grey-Box Attacks”. In: *Journal of Cryptology* 32.4 (2019), pp. 1095–1143. DOI: [10.1007/S00145-019-09315-1](https://doi.org/10.1007/S00145-019-09315-1).
- [Bos+16] Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen. “Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough”. In: *Proceedings of the 18th International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Ed. by Benedikt Gierlichs and Axel Y. Poschmann. Vol. 9813. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Aug. 2016, pp. 215–236. DOI: [10.1007/978-3-662-53140-2\\_11](https://doi.org/10.1007/978-3-662-53140-2_11).
- [Bou14] Guillaume Bouffard. “A Generic Approach for Protecting Java Card Smart Card Against Software Attacks”. PhD thesis. Limoges, France: Université de Limoges, Oct. 2014.
- [BG18] Guillaume Bouffard and Léo Gaspard. “Hardening a Java Card Virtual Machine Implementation with the MPU”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2018.

- [BGG21] Guillaume Bouffard, Vincent Giraud, and Léo Gaspard. “Java Card Virtual Machine Memory Organization: a Design Proposal”. In: *CoRR* (2021). arXiv: [2110.10037](https://arxiv.org/abs/2110.10037).
- [BIL11] Guillaume Bouffard, Julien Iguchi-Cartigny, and Jean-Louis Lanet. “Combined Software and Hardware Attacks on the Java Card Control Flow”. In: *Proceedings of the 10th International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Emmanuel Prouff. Vol. 7079. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Sept. 2011, pp. 283–296. doi: [10.1007/978-3-642-27257-8\\_18](https://doi.org/10.1007/978-3-642-27257-8_18).
- [Bou+13a] Guillaume Bouffard, Tom Khefif, Jean-Louis Lanet, Ismael Kane, and Sergio Casanova Salvia. “Accessing secure information using export file fraudulence”. In: *Proceedings of the International Conference on Risks and Security of Internet and Systems (CRiSIS)*. Ed. by Bruno Crispo, Ravi S. Sandhu, Nora Cuppens-Boulahia, Mauro Conti, and Jean-Louis Lanet. La Rochelle, France: IEEE Computer Society, Oct. 2013, pp. 1–5. doi: [10.1109/CRiSIS.2013.6766346](https://doi.org/10.1109/CRiSIS.2013.6766346).
- [Bou+14] Guillaume Bouffard, Michael Lackner, Jean-Louis Lanet, and Johannes Loinig. “Heap ... Hop! Heap Is Also Vulnerable”. In: *Proceedings of the 13th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Marc Joye and Amir Moradi. Vol. 8968. Lecture Notes in Computer Science. Paris, France: Springer, Nov. 2014, pp. 18–31. ISBN: 978-3-319-16762-6. doi: [10.1007/978-3-319-16763-3\\_2](https://doi.org/10.1007/978-3-319-16763-3_2).
- [BL12] Guillaume Bouffard and Jean-Louis Lanet. “The Next Smart Card Nightmare - Logical Attacks, Combined Attacks, Mutant Applications and Other Funny Things”. In: *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*. Ed. by David Naccache. Vol. 6805. Lecture Notes in Computer Science. Springer, 2012, pp. 405–424. ISBN: 978-3-642-28367-3. doi: [10.1007/978-3-642-28368-0\\_26](https://doi.org/10.1007/978-3-642-28368-0_26).
- [BL14a] Guillaume Bouffard and Jean-Louis Lanet. “Escalade de privilège dans une carte à puce Java Card”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2014.
- [BL14b] Guillaume Bouffard and Jean-Louis Lanet. “Reversing the operating system of a Java based smart card”. In: *Journal of Computer Virology and Hacking Techniques* 10.4 (July 2014), pp. 239–253. doi: [10.1007/s11416-014-0218-7](https://doi.org/10.1007/s11416-014-0218-7).
- [BL15] Guillaume Bouffard and Jean-Louis Lanet. “The ultimate control flow transfer in a Java based smart card”. In: *Computers and Security* 50 (May 2015), pp. 33–46. doi: [10.1016/j.cose.2015.01.004](https://doi.org/10.1016/j.cose.2015.01.004).
- [Bou+11] Guillaume Bouffard, Jean-Louis Lanet, Jean-Baptiste Machemie, Jean-Yves Poichotte, and Jean-Philippe Wary. “Evaluation of the Ability to Transform SIM Applications into Hostile Applications”. In: *Proceedings of the 10th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Emmanuel Prouff. Vol. 7079. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Sept. 2011, pp. 1–17. doi: [10.1007/978-3-642-27257-8\\_1](https://doi.org/10.1007/978-3-642-27257-8_1).
- [Bou+13b] Guillaume Bouffard, Mathieu Lassale, Sergio Ona Domene, Hanan Tadmori, and Jean-Louis Lanet. “Intégration d’une politique de flot de contrôle dans un automate de sécurité”. In: *8ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d’Information (SAR-SSI)*. Mont de Marsan, France, Sept. 2013.

- [BTL13a] Guillaume Bouffard, Bhagyalekshmy N. Thampi, and Jean-Louis Lanet. “Detecting Laser Fault Injection for Smart Cards Using Security Automata”. In: *Proceedings of the International Symposium on Security in Computing and Communications (SSCC)*. Ed. by Sabu M. Thampi, Pradeep K. Atrey, Chun-I Fan, and Gregorio Martínez Pérez. Vol. 377. Communications in Computer and Information Science. Mysore, India: Springer, Aug. 2013, pp. 18–29. ISBN: 978-3-642-40575-4. DOI: [10.1007/978-3-642-40576-1\\_3](https://doi.org/10.1007/978-3-642-40576-1_3).
- [BTL13b] Guillaume Bouffard, Bhagyalekshmy N. Thampi, and Jean-Louis Lanet. “Vulnerability Analysis on Smart Cards Using Fault Tree”. In: *Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*. Ed. by Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaâniche. Vol. 8153. Lecture Notes in Computer Science. Toulouse, France: Springer, Sept. 2013, pp. 82–93. ISBN: 978-3-642-40792-5. DOI: [10.1007/978-3-642-40793-2\\_8](https://doi.org/10.1007/978-3-642-40793-2_8).
- [BTL14] Guillaume Bouffard, Bhagyalekshmy N. Thampi, and Jean-Louis Lanet. “Security automation to mitigate laser-based fault attacks on smart cards”. In: *International Journal of Trust Management in Computing and Communications (IJTMCC)* 2.2 (Sept. 2014), pp. 185–205. DOI: [10.1504/IJTMCC.2014.064158](https://doi.org/10.1504/IJTMCC.2014.064158).
- [Bro24] C. Scott Brown. *Here are the phone update policies from every major Android manufacturer*. Oct. 2024. URL: <https://www.androidauthority.com/phone-update-policies-1658633/> (visited on 07/07/2025).
- [Bur+17] Jan Burchard, Mael Gay, Ange-Salomé Messeng Ekosso, Jan Horáček, Bernd Becker, Tobias Schubert, Martin Kreuzer, and Ilia Polian. “AutoFault: Towards Automatic Construction of Algebraic Fault Attacks”. In: *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Taipei, Taiwan: IEEE Computer Society, Sept. 2017, pp. 65–72. DOI: [10.1109/FDTC.2017.13](https://doi.org/10.1109/FDTC.2017.13).
- [BV21] SGS Brightsight BV. *H1D3 Secure Microcontroller with Crypto Library v0.1.4*. NSCIB-CC-0228971-CR. Version 2. Nov. 2021. URL: <https://www.commoncriteriaportal.org/files/epfiles/NSCIB-CC-0228971-CR.pdf> (visited on 07/07/2025).
- [CFT14] Andrea Calvagna, Andrea Fornaia, and Emiliano Tramontana. “Combinatorial Interaction Testing of a Java Card Static Verifier”. In: *Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation (ICST)*. Cleveland, Ohio, USA: IEEE Computer Society, Mar. 2014, pp. 84–87. ISBN: 978-0-7695-5194-4. DOI: [10.1109/ICSTW.2014.10](https://doi.org/10.1109/ICSTW.2014.10).
- [CT13] Andrea Calvagna and Emiliano Tramontana. “Automated Conformance Testing of Java Virtual Machines”. In: *Proceedings of the 7th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*. Ed. by Leonard Barolli, Fatos Xhafa, Hsing-Chung Chen, Antonio Fernandez Gómez-Skarmeta, and Farooq Hussain. Taichung, Taiwan: IEEE Computer Society, July 2013, pp. 547–552. DOI: [10.1109/CISIS.2013.99](https://doi.org/10.1109/CISIS.2013.99).
- [Cam+18] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. “Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers”. In: *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. Toronto, ON, Canada: ACM, Oct. 2018, pp. 163–177. DOI: [10.1145/3243734.3243802](https://doi.org/10.1145/3243734.3243802).
- [Car07] CardLogix. *Gemalto Achieves Major Breakthrough in Security Technology with Java Card Highest Level of Certification*. Oct. 2007. URL: <https://www.cardlogix.com/industry-news/gemalto-major-breakthrough-security-technology-java-card-eal7-certification/> (visited on 07/07/2025).

- [Cas02] Ludovic Casset. “Development of an Embedded Verifier for Java Card Byte Code Using Formal Methods”. In: *Proceedings of the International Symposium of Formal Methods Europe (FME)*. Ed. by Lars-Henrik Eriksson and Peter A. Lindsay. Vol. 2391. Lecture Notes in Computer Science. Copenhagen, Denmark: Springer, July 2002, pp. 290–309. ISBN: 3-540-43928-5. doi: [10.1007/3-540-45614-7\\_17](https://doi.org/10.1007/3-540-45614-7_17).
- [Cay22] Pierre-Louis Cayrel. “Contributions to code-based cryptography”. Habilitation to conduct researches. Aug. 2022.
- [Cer+20] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. “SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society, 2020, pp. 1416–1432. doi: [10.1109/SP40000.2020.00061](https://doi.org/10.1109/SP40000.2020.00061).
- [CCH23] Thomas Chamelot, Damien Couroussé, and Karine Heydemann. “MAFIA: Protecting the Microarchitecture of Embedded Systems Against Fault Injection Attacks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.12 (May 2023), pp. 4555–4568. doi: [10.1109/TCAD.2023.3276507](https://doi.org/10.1109/TCAD.2023.3276507).
- [Che00] Zhiqun Chen. *Java Card technology for smart cards: architecture and programmer’s guide*. Addison-Wesley Professional, June 2000. ISBN: 0201703297.
- [CX23] Tinghung Chiu and Wenjie Xiong. “SoK: Fault Injection Attacks on Cryptosystems”. In: *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*. Toronto, Canada: ACM, Oct. 2023, pp. 64–72. doi: [10.1145/3623652.3623671](https://doi.org/10.1145/3623652.3623671).
- [Cho+02] Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. “A White-Box DES Implementation for DRM Applications”. In: *Proceedings of the Security and Privacy in Digital Rights Management (DRM)*. Ed. by Joan Feigenbaum. Vol. 2696. Lecture Notes in Computer Science. Washington, DC, USA: Springer, Nov. 2002, pp. 1–15. ISBN: 3-540-40410-4. doi: [10.1007/978-3-540-44993-5\\_1](https://doi.org/10.1007/978-3-540-44993-5_1).
- [CB19] Ludovic Claudepierre and Philippe Besnier. “Microcontroller Sensitivity to Fault-Injection Induced by Near-Field Electromagnetic Interference”. In: *Proceedings of the International Symposium on Electromagnetic Compatibility (EMC)*. Sapporo, Japan, June 2019, pp. 673–676. doi: [10.23919/EMCTokyo.2019.8893701](https://doi.org/10.23919/EMCTokyo.2019.8893701).
- [Cla+21] Ludovic Claudepierre, Pierre-Yves Péneau, Damien Hardy, and Erven Rohou. “TRAITOR: A Low-Cost Evaluation Platform for Multifault Injection”. In: *Proceedings of the 21rd International Symposium on Advanced Security on Software and Systems (ASSS)*. Ed. by Weizhi Meng and Li Li. Virtual Event, Hong Kong: ACM, June 2021, pp. 51–56. doi: [10.1145/3457340.3458303](https://doi.org/10.1145/3457340.3458303).
- [Com24] Common Criteria. *Certified Products*. 2024. URL: <https://www.commoncriteriaportal.org/products/index.cfm> (visited on 07/07/2025).
- [DB21] Jean Dubreuil and Guillaume Bouffard. “PhiAttack - Rewriting the Java Card Class Hierarchy”. In: *Proceedings of the 20th International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Lübeck, Germany: Springer, Nov. 2021, pp. 275–288. doi: [10.1007/978-3-030-97348-3\\_15](https://doi.org/10.1007/978-3-030-97348-3_15).

- [Dub+12] Jean Dubreuil, Guillaume Bouffard, Jean-Louis Lanet, and Julien Cartigny. “Type Classification against Fault Enabled Mutant in Java Based Smart Card”. In: *Proceedings of the 7th International Conference on Availability, Reliability and Security (ARES)*. Prague, Czech Republic: IEEE Computer Society, Aug. 2012, pp. 551–556. ISBN: 978-1-4673-2244-7. DOI: [10.1109/ARES.2012.24](https://doi.org/10.1109/ARES.2012.24).
- [Dub+13] Jean Dubreuil, Guillaume Bouffard, Bhagyalekshmy N. Thampi, and Jean-Louis Lanet. “Mitigating Type Confusion on Java Card”. In: *International Journal of Secure Software Engineering* 4.2 (2013), pp. 19–39. DOI: [10.4018/jsse.2013040102](https://doi.org/10.4018/jsse.2013040102).
- [DBP23] Soline Ducouso, Sébastien Bardin, and Marie-Laure Potet. “Adversarial Reachability for Program-level Security Analysis”. In: *Proceedings of the 32nd European Symposium on Programming, (ESOP) Held as Part of the European Joint Conferences on Theory and Practice of Software, (ETAPS)*. Ed. by Thomas Wies. Vol. 13990. Lecture Notes in Computer Science. Paris, France: Springer, Apr. 2023, pp. 59–89. DOI: [10.1007/978-3-031-30044-8\\_3](https://doi.org/10.1007/978-3-031-30044-8_3).
- [Dud19] Sébastien Dudek. “V2G Injector: Whispering to cars and charging units through the Power-Line”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 7, 2019.
- [DLM21] Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. “Modeling and Simulating Electromagnetic Fault Injection”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.4 (2021), pp. 680–693. DOI: [10.1109/TCAD.2020.3003287](https://doi.org/10.1109/TCAD.2020.3003287).
- [Dur+16] Louis Dureuil, Guillaume Petiot, Marie-Laure Potet, Thanh-Ha Le, Aude Crohen, and Philippe de Choudens. “FISSC: A Fault Injection and Simulation Secure Collection”. In: *Proceedings of the 35th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*. Ed. by Amund Skavhaug, Jérémie Guiochet, and Friedemann Bitsch. Vol. 9922. Lecture Notes in Computer Science. Trondheim, Norway: Springer, Sept. 2016, pp. 3–11. ISBN: 978-3-319-45476-4. DOI: [10.1007/978-3-319-45477-1\\_1](https://doi.org/10.1007/978-3-319-45477-1_1).
- [EMV07] EMV. *EMV Card Personalisation Specification*. July 2007.
- [EMV08] EMV. *Common Payment Application Specification*. Mar. 2008.
- [EMV11a] EMV. *Book 1. Integrated Circuit Card Specifications for Payment Systems*. Version 4.3. Geneva, Switzerland, Nov. 2011.
- [EMV11b] EMV. *Book 2. Security and Key Management*. Version 4.3. Nov. 2011.
- [EMV11c] EMV. *Book 3. Application Specification*. Version 4.3. Nov. 2011.
- [EMV11d] EMV. *Book 4. Cardholder, Attendant, and Acquirer Interface Requirements*. Version 4.3. Nov. 2011.
- [Eur14] Eurosmart. *Smartcard IC Platform Protection Profile with Augmentation Packages*. BSI-CC-PP-0084. Version 1.0. Jan. 2014. URL: [https://www.commoncriteriaportal.org/files/ppfiles/pp0084b\\_pdf.pdf](https://www.commoncriteriaportal.org/files/ppfiles/pp0084b_pdf.pdf) (visited on 07/07/2025).
- [Eur22] Eurosmart. *Secure Sub-System in System-on-Chip Protection Profile*. BSI-CC-PP-0117. Version 1.5. Mar. 2022. URL: [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte/ReportePP/pp0117a\\_pdf](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte/ReportePP/pp0117a_pdf) (visited on 07/07/2025).
- [Evd24] Mikhail Evdokimov. “0-click RCE on the IVI component: Pwn2Own Automotive edition”. In: *Hexacon* (Oct. 2024).

- [Fan+22] Clément Fanjas, Clément Gaine, Driss Aboukassimi, Simon Pontié, and Olivier Potin. “Combined Fault Injection and Real-Time Side-Channel Analysis for Android Secure-Boot Bypassing”. In: *Proceedings of the 21st International Conference of Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Ileana Buhan and Tobias Schneider. Vol. 13820. Lecture Notes in Computer Science. Birmingham, UK: Springer, Nov. 2022, pp. 25–44. doi: [10.1007/978-3-031-25319-5\\_2](https://doi.org/10.1007/978-3-031-25319-5_2).
- [FV10] Emilie Faugeron and Sebastien Valette. “How to hoax an off-card verifier”. In: *e-smart* (Sept. 2010).
- [Gai+20] Clément Gaine, Driss Aboukassimi, Simon Pontié, Jean-Pierre Nikolovski, and Jean-Max Dutertre. “Electromagnetic Fault Injection as a New Forensic Approach for SoCs”. In: *Proceedings of the 12th IEEE International Workshop on Information Forensics and Security (WIFS)*. New York City, New York, USA: IEEE Computer Society, Dec. 2020, pp. 1–6. ISBN: 978-1-7281-9930-6. doi: [10.1109/WIFS49906.2020.9360902](https://doi.org/10.1109/WIFS49906.2020.9360902).
- [Gas17] Léo Gaspard. “Implementation of a Secure Operating System for Java Card-based Secure Element”. Master’s thesis. Palaiseau, France: École Polytechnique, Sept. 2017.
- [Gir19] Vincent Giraud. “Secure Implementation of GlobalPlatform for Java Card Platform”. Master’s thesis. Rennes, France: INSA, Sept. 2019.
- [Gir23a] Vincent Giraud. “Method for protecting a terminal against a side channel attack”. FR3142818A1. Oct. 2023. URL: <https://patents.google.com/patent/FR3142818A1>.
- [Gir23b] Vincent Giraud. “White box encoding”. WO2024083849A1. Oct. 2023. URL: <https://patents.google.com/patent/WO2024083849A1>.
- [Gir23c] Vincent Giraud. “White-box cryptographic keys”. WO2024083855A1. Oct. 2023. URL: <https://patents.google.com/patent/WO2024083855A1>.
- [Gir24] Vincent Giraud. “Application security on uncontrolled systems. Study of the risks, protections, stakes and interests around trust in off-the-shelf computer products”. PhD thesis. Paris, France: École Normale Supérieure, Sept. 2024.
- [GB23] Vincent Giraud and Guillaume Bouffard. “Faulting original McEliece’s implementations is possible. How to mitigate this risk?” In: *IEEE European Workshops on Symposium on Security and Privacy (EuroS&PW)*. Delft, Netherlands: IEEE, July 2023, pp. 311–319. doi: [10.1109/EuroSPW59978.2023.00039](https://doi.org/10.1109/EuroSPW59978.2023.00039).
- [GN23a] Vincent Giraud and David Naccache. “Batterie à bord: quand les jauges de carburant dépassent les limites”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2023.
- [GN23b] Vincent Giraud and David Naccache. “Power Analysis Pushed too Far: Breaking Android-Based Isolation with Fuel Gauges”. In: *Proceedings of the 18th International Workshop on Security (IWSEC)*. Ed. by Junji Shikata and Hiroki Kuzuno. Vol. 14128. Lecture Notes in Computer Science. Yokohama, Japan: Springer, Aug. 2023, pp. 3–15. doi: [10.1007/978-3-031-41326-1\\_1](https://doi.org/10.1007/978-3-031-41326-1_1).
- [Glo12] Global Platform. *Java Card Contactless API and Export File for Card Specification v2.2.1 (org.globalplatform.contactless) v1.1*. Feb. 2012. URL: <https://globalplatform.org/specs-library/java-card-contactless-api-ad-export-file-for-card-specification-v2-2-1-org-globalplatform-contactless-v1-1/> (visited on 07/07/2025).
- [Glo18a] GlobalPlatform. *Introduction to Secure Elements*. May 2018. URL: <https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Secure-Element-15May2018.pdf> (visited on 07/07/2025).

- [Glo18b] GlobalPlatform. *Root of Trust Definitions and Requirements*. Version 1.1. June 2018. URL: [https://globalplatform.org/wp-content/uploads/2018/07/GP\\_RoT\\_Definitions\\_and\\_Requirements\\_v1.1\\_PublicRelease-2018-06-28.pdf](https://globalplatform.org/wp-content/uploads/2018/07/GP_RoT_Definitions_and_Requirements_v1.1_PublicRelease-2018-06-28.pdf).
- [Glo19] GlobalPlatform. *An Introduction to GlobalPlatform's Device Trust Architecture*. July 2019.
- [Glo20a] GlobalPlatform. *TEE Client Core API Specification*. GPD\_SPE\_007. Version 1.0. July 2020. URL: [https://globalplatform.org/wp-content/uploads/2010/07/TEE\\_Client\\_API\\_Specification-V1.0.pdf](https://globalplatform.org/wp-content/uploads/2010/07/TEE_Client_API_Specification-V1.0.pdf) (visited on 07/07/2025).
- [Glo20b] GlobalPlatform. *TEE Protection Profile*. GPD\_SPE\_021. Version 1.3. July 2020. URL: <https://globalplatform.org/specs-library/tee-protection-profile-v1-3/> (visited on 07/07/2025).
- [Glo21] GlobalPlatform. *TEE Internal Core API Specification*. Version 1.3.1. July 2021. URL: [https://globalplatform.org/wp-content/uploads/2021/03/GPD\\_TEE\\_Internal\\_Core\\_API\\_Specification\\_v1.3.1\\_PublicRelease\\_CC.pdf](https://globalplatform.org/wp-content/uploads/2021/03/GPD_TEE_Internal_Core_API_Specification_v1.3.1_PublicRelease_CC.pdf) (visited on 07/07/2025).
- [Glo23] GlobalPlatform. *Trust & Security in Automotive Systems*. Tech. rep. Oct. 2023. URL: [https://globalplatform.org/wp-content/uploads/2023/10/GP-Trust-for-Secure-AutoServices-White-Paper\\_Web\\_Spreads.pdf](https://globalplatform.org/wp-content/uploads/2023/10/GP-Trust-for-Secure-AutoServices-White-Paper_Web_Spreads.pdf) (visited on 07/07/2025).
- [Gon26] Gwenn Le Gonidec. "Securing RISC-V System-on-Chip against Energy-based Attacks". PhD thesis. Rennes, France: Université Bretagne Sud, 2026. In preparation.
- [Gon+25] Gwenn Le Gonidec, Guillaume Bouffard, Jean-Christophe Prévotet, and Maria Méndez Real. "Do Not Trust Power Management: A Survey on Internal Energy-based Attacks Circumventing Trusted Execution Environments Security Properties". In: *ACM Transactions on Embedded Computing Systems* 24.4 (July 2025). ISSN: 1539-9087. DOI: [10.1145/3735556](https://doi.org/10.1145/3735556).
- [Hal+08] Daniel Halperin, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H. Maisel. "Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. Oakland, California, USA: IEEE Computer Society, May 2008, pp. 129–142. DOI: [10.1109/SP.2008.31](https://doi.org/10.1109/SP.2008.31).
- [Ham+12] Samiya Hamadouche, Guillaume Bouffard, Jean-Louis Lanet, Bruno Dorsemayne, Bastien Nouhant, Alexandre Magloire, and Arnaud Reygnaud. "Subverting Byte Code Linker service to characterize Java Card API". In: *Proceedings of the 7th Conference on Network and Information Systems Security (SAR-SSI)*. Cabourg, France, May 2012, pp. 75–81. ISBN: 978-2-9542630-0-7.
- [Heu24] Maurice Heumann. *Bypassing Denuvo in Hogwarts Legacy*. Mar. 2024. URL: <https://momo5502.com/posts/2024-03-31-bypassing-denuvo-in-hogwarts-legacy/> (visited on 07/07/2025).
- [HSP20] Max Hoffmann, Falk Schellenberg, and Christof Paar. "ARMORY: Fully Automated and Exhaustive Fault Simulation on ARM-M Binaries". In: *IEEE Transactions on Information Forensics and Security* 16 (Sept. 2020), pp. 1058–1073. DOI: [10.1109/TIFS.2020.3027143](https://doi.org/10.1109/TIFS.2020.3027143).
- [Idr+17] Noredine El Janati El Idrissi, Guillaume Bouffard, Jean-Louis Lanet, and Said El Hajji. "Trust can be misplaced". In: *Journal of Cryptographic Engineering* 7.1 (2017), pp. 21–34. DOI: [10.1007/s13389-016-0142-5](https://doi.org/10.1007/s13389-016-0142-5).
- [Int22a] Intel. *Intel™ Software Guard Extensions (Intel™ SGX)*. 2022. URL: <https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html> (visited on 07/07/2025).

- [Int21] International Civil Aviation Organization. *Machine Readable Travel Documents. Part 11: Security Mechanisms for MRTDs*. Version Eighth Edition. 2021.
- [Int06] International Organization for Standardization. *ISO/IEC 7816. Part 3: Cards with contacts—Electrical interface and transmission protocols*. Geneva, Switzerland, 2006.
- [Int13] International Organization for Standardization. *ISO/IEC 7816. Part 4: Organization, security and commands for interchange*. Geneva, Switzerland, 2013.
- [Int15] International Organization for Standardization. *ISO/IEC 11889 Information Technology — Trusted Platform Module Library. Part 1: Architecture*. Geneva, Switzerland, 2015.
- [Int18] International Organization for Standardization. *ISO/IEC 14443 Cards and security devices for personal identification — Contactless proximity objects. Part 4: Transmission protocol*. Geneva, Switzerland, 2018.
- [Int22b] International Organization for Standardization. *ISO/IEC 15408: Information security, cybersecurity and privacy protection. Evaluation criteria for IT security*. Geneva, Switzerland, 2022.
- [ITB23] Alexandre Iooss, Thomas Troughkine, and Guillaume Bouffard. “Pew Pew, I’m root! De la caractérisation à l’exploitation: un voyage plein d’embûches”. fr. In: *Journée thématique sur les attaques par injection de fautes (JAIF)* (Sept. 2023). URL: <https://jaif.io/2023/>.
- [Jac+23] Hans Niklas Jacob, Christian Werling, Robert Bühren, and Jean -Pierre Seifert. “faultPM: Exposing AMD fTPMs’ Deepest Secrets”. In: *Proceedings of the 8th IEEE European Symposium on Security and Privacy*. Delft, Netherlands: IEEE Computer Society, July 2023, pp. 1128–1142. doi: [10.1109/EUROSP57164.2023.00069](https://doi.org/10.1109/EUROSP57164.2023.00069).
- [JM14] Marc Joye and Amir Moradi, eds. *Proceedings of the 13th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Vol. 8968. Lecture Notes in Computer Science. Paris, France: Springer, Nov. 2014. ISBN: 978-3-319-16762-6. doi: [10.1007/978-3-319-16763-3](https://doi.org/10.1007/978-3-319-16763-3).
- [KAS20] J. Karthik, P. P. Amritha, and M. Sethumadhavan. “Video Game DRM: Analysis and Paradigm Solution”. In: *Proceedings of the 11th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2020, Kharagpur, India, July 1-3, 2020*. IEEE Computer Society, 2020, pp. 1–4. doi: [10.1109/ICCCNT49239.2020.9225560](https://doi.org/10.1109/ICCCNT49239.2020.9225560).
- [Kas23] Kaspersky. *Kaspersky discloses iPhone hardware feature vital in Operation Triangulation case*. Dec. 2023. URL: <https://www.kaspersky.com/about/press-releases/kaspersky-discloses-iphone-hardware-feature-vital-in-operation-triangulation-case> (visited on 07/07/2025).
- [Kha+16] Rafiullah Khan, Peter Maynard, Kieran McLaughlin, David M. Lavery, and Sakir Sezer. “Threat Analysis of BlackEnergy Malware for Synchronphasor based Real-time Control and Monitoring in Smart Grid”. In: *Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016, ICS-CSR*. Ed. by Thomas Brandstetter and Helge Janicke. Workshops in Computing. Queen’s Belfast University, UK: BCS, Aug. 2016. URL: <https://ewic.bcs.org/content/ConWebDoc/56478> (visited on 07/07/2025).
- [Kin24] Beth Kindig. *Arm Stock: AI Chip Favorite Is Overpriced*. Mar. 2024. URL: <https://www.forbes.com/sites/bethkindig/2024/03/21/arm-stock-ai-chip-favorite-is-overpriced/> (visited on 07/07/2025).

- [Koc+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. “Spectre Attacks: Exploiting Speculative Execution”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE, May 2019, pp. 1–19. doi: [10.1109/SP.2019.00002](https://doi.org/10.1109/SP.2019.00002).
- [KH14] Thomas Korak and Michael Hoefler. “On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms”. In: *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Ed. by Assia Tria and Dooho Choi. Busan, South Korea: IEEE Computer Society, Sept. 2014, pp. 8–17. ISBN: 978-1-4799-6292-1. doi: [10.1109/FDTC.2014.11](https://doi.org/10.1109/FDTC.2014.11).
- [Lab24] CyberThreat Research Lab. *Under Pressure: Exploring a Zero-Click RCE Vulnerability in Tesla’s TPMS*. Dec. 2024. URL: <https://vicone.com/blog/under-pressure-exploring-a-zero-click-rce-vulnerability-in-teslas-tpms> (visited on 07/07/2025).
- [Lan11a] Julien Lancia. “Un framework de fuzzing pour cartes à puce: application aux protocoles EMV”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2011.
- [LB15] Julien Lancia and Guillaume Bouffard. “Java Card Virtual Machine Compromising from a Bytecode Verified Applet”. In: *Proceedings of the 14th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Vol. 9514. Lecture Notes in Computer Science. Bochum, Germany: Springer, Nov. 2015, pp. 75–88. doi: [10.1007/978-3-319-31271-2\\_5](https://doi.org/10.1007/978-3-319-31271-2_5).
- [LB16] Julien Lancia and Guillaume Bouffard. “Fuzzing and Overflows in Java Card Smart Cards”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2016.
- [Lan+18] Jean-Louis Lanet, Hélène Le Boudier, Mohammed Benattou, and Axel Legay. “When time meets test”. In: *International Journal of Information Security* 17.4 (2018), pp. 395–409. doi: [10.1007/s10207-017-0371-3](https://doi.org/10.1007/s10207-017-0371-3).
- [Lan+14] Jean-Louis Lanet, Guillaume Bouffard, Rokia Lamrani, Ranim Chakra, Afef Mestiri, Mohammed Monsif, and Abdellatif Fandi. “Memory Forensics of a Java Card Dump”. In: *Proceedings of the 13th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Marc Joye and Amir Moradi. Vol. 8968. Lecture Notes in Computer Science. Paris, France: Springer, Nov. 2014, pp. 3–17. ISBN: 978-3-319-16762-6. doi: [10.1007/978-3-319-16763-3\\_1](https://doi.org/10.1007/978-3-319-16763-3_1).
- [Lan11b] Ralph Langner. “Stuxnet: Dissecting a Cyberwarfare Weapon”. In: *IEEE Security & Privacy* 9.3 (May 2011), pp. 49–51. doi: [10.1109/MSP.2011.67](https://doi.org/10.1109/MSP.2011.67).
- [Lap+20] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. “Browser Fingerprinting: A Survey”. In: *ACM Transactions on the Web* 14.2 (2020), 8:1–8:33. doi: [10.1145/3386040](https://doi.org/10.1145/3386040).
- [Lau+19] Johan Laurent, Vincent Berouille, Christophe Deleuze, Florian Pebay-Peyroula, and Athanasios Papadimitriou. “Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a RISC-V processor”. In: *Microprocessors and Microsystems* 71 (Aug. 2019). doi: [10.1016/J.MICPRO.2019.102862](https://doi.org/10.1016/J.MICPRO.2019.102862).
- [Lau+21] Johan Laurent, Christophe Deleuze, Florian Pebay-Peyroula, and Vincent Berouille. “Bridging the Gap between RTL and Software Fault Injection”. In: *ACM Journal on Emerging Technologies in Computing Systems* 17.3 (May 2021), 38:1–38:24. doi: [10.1145/3446214](https://doi.org/10.1145/3446214).
- [Law10] Nate Lawson. *How the PS3 Hypervisor was Hacked*. Jan. 2010. URL: <https://rdist.root.org/2010/01/27/how-the-ps3-hypervisor-was-hacked/> (visited on 07/07/2025).

- [Lee+20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. “Keystone: an open framework for architecting trusted execution environments”. In: *Proceedings of the 15th EuroSys Conference*. Ed. by Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer. Heraklion, Greece: ACM, Apr. 2020, pp. 1–16. doi: [10.1145/3342195.3387532](https://doi.org/10.1145/3342195.3387532).
- [Leg23] Alessandro Legnani. *Trusted Firmware for a Research Computer*. Aug. 2023. doi: [20.500.11850/634201](https://doi.org/20.500.11850/634201).
- [Ler02] Xavier Leroy. “Bytecode verification on Java smart cards”. In: *Software - Practice and Experience (SPE)* 32.4 (2002), pp. 319–340. doi: [10.1002/spe.438](https://doi.org/10.1002/spe.438).
- [LG19] Hao hao Liao and Catherine H. Gebotys. “Methodology for EM Fault Injection: Charge-based Fault Model”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Ed. by Jürgen Teich and Franco Fummi. Florence, Italy: IEEE, Mar. 2019, pp. 256–259. doi: [10.23919/DATE.2019.8715150](https://doi.org/10.23919/DATE.2019.8715150).
- [Lip+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. “Meltdown: Reading Kernel Memory from User Space”. In: *Proceedings of the 27th USENIX Security Symposium*. Ed. by William Enck and Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, Aug. 2018, pp. 973–990.
- [Lon+15] Jake Longo, Elke De Mulder, Dan Page, and Michael Tunstall. “SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip”. In: *Proceedings of the 17th International Workshop Cryptographic Hardware and Embedded Systems (CHES)*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Saint-Malo, France: Springer, Sept. 2015, pp. 620–640. doi: [10.1007/978-3-662-48324-4\\_31](https://doi.org/10.1007/978-3-662-48324-4_31).
- [Low24] LowRISC. *OpenTitan*. 2024. URL: <https://github.com/lowRISC/opentitan> (visited on 07/07/2025).
- [MBB16] Fabien Majéric, Eric Bourbao, and Lilian Bossuet. “Electromagnetic security tests for SoC”. In: *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. Monte Carlo, Monaco: IEEE Computer Society, Dec. 2016, pp. 265–268. ISBN: 978-1-5090-6113-6. doi: [10.1109/ICECS.2016.7841183](https://doi.org/10.1109/ICECS.2016.7841183).
- [Mal22] Louisa Malki. “Fingerprinting of Embedded Software Implementation”. Master’s thesis. Paris, France: École 42, Sept. 2022.
- [Man+18] Heiko Mantel, Johannes Schickel, Alexandra Weber, and Friedrich Weber. “How Secure Is Green IT? The Case of Software-Based Energy Side Channels”. In: *Proceedings of the 23rd European Symposium on Research in Computer Security (ESORICS)*. Ed. by Javier López, Jianying Zhou, and Miguel Soriano. Vol. 11098. Lecture Notes in Computer Science. Barcelona, Spain: Springer, Sept. 2018, pp. 218–239. doi: [10.1007/978-3-319-99073-6\\_11](https://doi.org/10.1007/978-3-319-99073-6_11).
- [Mar25] Amélie Marotta. “Effects of synchronous clock glitch on the security of integrated circuits”. PhD thesis. Rennes, France: Université de Rennes, June 2025.
- [Mar+24] Amélie Marotta, Ronan Lashermes, Guillaume Bouffard, Olivier Sentieys, and Rachid Dafali. “Characterizing and Modeling Synchronous Clock-Glitch Fault Injection”. In: *Proceedings of the 15th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*. Ed. by Romain Wacquez and Naofumi Homma. Vol. 14595. Lecture Notes in Computer Science. Gardanne, France: Springer, Apr. 2024, pp. 3–21. doi: [10.1007/978-3-031-57543-3\\_1](https://doi.org/10.1007/978-3-031-57543-3_1).
- [Mel24] Willem Melching. *Bypassing the Renesas RH850/P1M-E read protection using fault injection*. Nov. 8, 2024. URL: <https://icanhack.nl/blog/rh850-glitch/> (visited on 07/07/2025).

- [MV13] Charlie Miller and Chris Valasek. “Remote Exploitation of an Unaltered Passenger Vehicle”. In: *DEF CON 2023* (Aug. 2013).
- [Nat17] National Information Assurance Partnership. *Protection Profile for Mobile Device Fundamentals*. Version 3.1. June 2017. URL: [https://www.commoncriteriaportal.org/files/ppfiles/pp\\_md\\_v3.1.pdf](https://www.commoncriteriaportal.org/files/ppfiles/pp_md_v3.1.pdf) (visited on 07/07/2025).
- [Nat21a] National Information Assurance Partnership. *Protection Profile for Application Software*. CCEVS-VR-PP-0080. Version 1.4. Oct. 2021. URL: [https://www.commoncriteriaportal.org/files/ppfiles/PP\\_APP\\_V1.4\\_VR.pdf](https://www.commoncriteriaportal.org/files/ppfiles/PP_APP_V1.4_VR.pdf) (visited on 07/07/2025).
- [Nat21b] National Information Assurance Partnership. *Validation Report. Google Pixel Phones on Android 11.0*. CCEVS-VR-11124-2021. Version 1.0. Aug. 2021. URL: [https://www.commoncriteriaportal.org/files/epfiles/st\\_vid11124-vr.pdf](https://www.commoncriteriaportal.org/files/epfiles/st_vid11124-vr.pdf) (visited on 07/07/2025).
- [Nem+17] Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. “The Return of Coppersmith’s Attack: Practical Factorization of Widely Used RSA Moduli”. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. Dallas, TX, USA: ACM, Oct. 2017, pp. 1631–1648. DOI: [10.1145/3133956.3133969](https://doi.org/10.1145/3133956.3133969).
- [NLn24] NLnet. *NaxRiscv*. 2024. URL: <https://github.com/SpinalHDL/NaxRiscv> (visited on 07/07/2025).
- [Nou+09] Agnès Cristèle Noubissi, Ahmadou Al Khary Sere, Julien Iguchi-Cartigny, Jean-Louis Lanet, Guillaume Bouffard, and Julien Boutet. “Carte à puce : Attaques et Contremesures”. In: *Majestic*. 1112. Avignon, France, Nov. 2009.
- [OFI20] Colin O’Flynn. “BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks”. In: *IACR Cryptology ePrint Archive* (2020), p. 937. [eprint](https://eprint.iacr.org/2020/937).
- [Ope24a] OpenHW Group. *CVA6 RISC-V CPU*. 2024. URL: <https://github.com/openhwgroup/cva6> (visited on 07/07/2025).
- [Ope24b] OpenHW Group. *OpenHW Group CORE-V CV32E40S RISC-V IP*. 2024. URL: <https://github.com/openhwgroup/cv32e40s> (visited on 07/07/2025).
- [Ope23] OpenTitan Developer Team. *OpenTitan’s use cases*. 2023. URL: [https://opentitan.org/book/doc/use\\_cases/](https://opentitan.org/book/doc/use_cases/) (visited on 07/07/2025).
- [Ora21a] Oracle. *Java Card 3 Platform, Runtime Environment Specification. Classic Edition*. Version 3.1. Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065: Oracle, Feb. 2021.
- [Ora21b] Oracle. *Java Card 3 Platform, Virtual Machine Specification. Classic Edition*. Version 3.1. Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065: Oracle, Feb. 2021.
- [Ora21c] Oracle. *Java Card Protection Profile – Open Configuration*. Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, May 2021.
- [Pas22] Calinel Pasteanu. *Oracle Celebrates the Java Card Forum’s 25th Anniversary*. Oct. 2022. URL: <https://blogs.oracle.com/java/post/java-card-forum-25-years-anniversary> (visited on 07/07/2025).
- [PSF22] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. “Exploring Widevine for Fun and Profit”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society, May 2022, pp. 277–288. DOI: [10.1109/SPW54247.2022.9833867](https://doi.org/10.1109/SPW54247.2022.9833867).
- [Pay19] Payment Card Industry. *Contactless Payments on COTS (CPoC™)*. Version 1.0. Dec. 2019. URL: [https://listings.pcisecuritystandards.org/documents/Contactless\\_Payments\\_on\\_COTS-Security\\_and\\_Test\\_Requirements-v1.0.pdf](https://listings.pcisecuritystandards.org/documents/Contactless_Payments_on_COTS-Security_and_Test_Requirements-v1.0.pdf) (visited on 07/07/2025).

- [Pay20] Payment Card Industry. *Software-based PIN Entry on COTS (SPoC)*. June 2020. URL: <https://www.pcisecuritystandards.org/standards/software-based-pin-entry-on-cots-spoc/> (visited on 07/07/2025).
- [Pet+15] Martin Petrvalsky, Tania Richmond, Milos Drutarovsky, Pierre-Louis Cayrel, and Viktor Fischer. "Countermeasure against the SPA attack on an embedded McEliece cryptosystem". In: *Proceedings of 25th IEEE International Conference Radioelektronika (MAREW)*. Pardubice, Czech Republic, Apr. 2015, pp. 462–466. doi: [10.1109/RADIOELEK.2015.7129055](https://doi.org/10.1109/RADIOELEK.2015.7129055).
- [Pic+23] Stjepan Picsek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. "SoK: Deep Learning-based Physical Side-channel Analysis". In: *ACM Computing Surveys* 55.11 (2023), 227:1–227:35. doi: [10.1145/3569577](https://doi.org/10.1145/3569577).
- [Pot+14] Marie-Laure Potet, Laurent Mounier, Maxime Puys, and Louis Dureuil. "Lazart: A Symbolic Approach for Evaluation the Robustness of Secured Codes against Control Flow Injections". In: *Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation, (ICST)*. Cleveland, Ohio, USA: IEEE Computer Society, Mar. 2014, pp. 213–222. doi: [10.1109/ICST.2014.34](https://doi.org/10.1109/ICST.2014.34).
- [Pro+19] Julien Proy, Karine Heydemann, Alexandre Berzati, Fabien Majéric, and Albert Cohen. "A First ISA-Level Characterization of EM Pulse Effects on Superscalar Microarchitectures: A Secure Software Perspective". In: *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES)*. Canterbury, UK: ACM, Aug. 2019, 7:1–7:10. doi: [10.1145/3339252.3339253](https://doi.org/10.1145/3339252.3339253).
- [Ram23] Rambus Press. *Hardware Root of Trust: Everything you need to know*. July 2023. URL: <https://www.rambus.com/blogs/hardware-root-of-trust/> (visited on 07/07/2025).
- [Rao22] Anil Rao. *Rising to the Challenge. Data Security with Intel Confidential Computing*. Jan. 2022. URL: <https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141> (visited on 07/07/2025).
- [RBL12] Tiana Razafindralambo, Guillaume Bouffard, and Jean-Louis Lanet. "A Friendly Framework for Hidding fault enabled virus for Java Based Smartcard". In: *Proceedings of the 26th Annual Conference on Data and Applications Security and Privacy (DBSec)*. Ed. by Nora Cuppens-Boulahia, Frédéric Cuppens, and Joaquín García-Alfaro. Vol. 7371. Lecture Notes in Computer Science. Paris, France: Springer, July 2012, pp. 122–128. doi: [10.1007/978-3-642-31540-4\\_10](https://doi.org/10.1007/978-3-642-31540-4_10).
- [Raz+12] Tiana Razafindralambo, Guillaume Bouffard, Bhagyalekshmy N. Thampi, and Jean-Louis Lanet. "A Dynamic Syntax Interpretation for Java Based Smart Card to Mitigate Logical Attacks". In: *Proceedings of the International Conference on Recent Trends in Computer Networks and Distributed Systems Security (SNDS)*. Trivandrum, India, Oct. 2012, pp. 185–194. doi: [10.1007/978-3-642-34135-9\\_19](https://doi.org/10.1007/978-3-642-34135-9_19).
- [Raz11] Razkar. *The Reset Glitch Hack - A New Exploit on Xbox 360*. Aug. 2011. URL: <https://www.logic-sunrise.com/news-341321-the-reset-glitch-hack-a-new-exploit-on-xbox-360-en.html> (visited on 07/07/2025).
- [Ric+21] Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. "FIVER - Robust Verification of Countermeasures against Fault Injections". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.4 (Sept. 2021), pp. 447–473. doi: [10.46586/TCHES.V2021.I4.447-473](https://doi.org/10.46586/TCHES.V2021.I4.447-473).

- [Riv+15] Lionel Rivière, Zakaria Najm, Pablo Rauzy, Jean- Luc Danger, Julien Bringer, and Laurent Sauvage. “High precision fault injections on the instruction cache of ARMv7-M architectures”. In: *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. Washington, District of Columbia, USA: IEEE Computer Society, May 2015, pp. 62–67. ISBN: 978-1-4673-7420-0. DOI: [10.1109/HST.2015.7140238](https://doi.org/10.1109/HST.2015.7140238).
- [Roc+21] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. “A Side Journey To Titan”. In: *Proceedings of the 30th USENIX Security Symposium*. Ed. by Michael D. Bailey and Rachel Greenstadt. USENIX Association, Aug. 2021, pp. 231–248.
- [Ros21] Ever Atilano Rosales. “Security of the Secure Boot against Fault Attacks”. Master’s thesis. Paris, France: Université Paris 6, Sept. 2021.
- [Rot24] Thomas Roth. “ACE Up the Sleeve: Hacking Into Apple’s New USB-C Controller”. In: (Dec. 2024). URL: <https://media.ccc.de/v/38c3-ace-up-the-sleeve-hacking-into-apple-s-new-usb-c-controller> (visited on 07/07/2025).
- [Rou+19] Gauvain Tanguy Henri Gabriel Isidore Roussel-Tarbouriech, Noel Menard, Tyler True, TiniVi, and Reisyukaku. “Methodically Defeating Nintendo Switch Security”. In: *CoRR* abs/1905.07643 (June 2019). arXiv: [1905.07643](https://arxiv.org/abs/1905.07643).
- [SAM24] SAMSUNG Electronics Co. Ltd. *STRONGV4P00 of S5E9945 with Specific IC Dedicated Software*. NSCIB-CC-2300085-02. Version 1.0. Mar. 2024. URL: [https://www.commoncriteriaportal.org/nfs/ccpfiles/files/epfiles/NSCIB-CC-2300085-01-ST\\_Lite\\_v0.1.pdf](https://www.commoncriteriaportal.org/nfs/ccpfiles/files/epfiles/NSCIB-CC-2300085-01-ST_Lite_v0.1.pdf) (visited on 07/07/2025).
- [SFL13] Aymerick Savary, Marc Frappier, and Jean-Louis Lanet. “Detecting Vulnerabilities in Java-Card Bytecode Verifiers Using Model-Based Testing”. In: *Proceedings of the 10th International Conference on Integrated Formal Methods (IFM)*. Ed. by Einar Broch Johnsen and Luigia Petre. Vol. 7940. Lecture Notes in Computer Science. Turku, Finland: Springer, June 2013, pp. 223–237. ISBN: 978-3-642-38612-1. DOI: [10.1007/978-3-642-38613-8\\_16](https://doi.org/10.1007/978-3-642-38613-8_16).
- [Sch+24] S. van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, D. Genkin, A. Miller, E. Ronen, Y. Yarom, and C. Garman. “SoK: SGX.Fail: How Stuff Gets eXposed”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 248–248. DOI: [10.1109/SP54263.2024.00260](https://doi.org/10.1109/SP54263.2024.00260).
- [Sch+23a] Pasquale Davide Schiavone, Simone Machetti, Miguel Peón Quirós, Jose Miranda, Benoît W. Denkinger, Thomas Christoph Müller, Ruben Rodríguez, Saverio Nasturzio, and David Atienza Alonso. “X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller”. In: *Proceedings of the 20th ACM International Conference on Computing Frontiers (CF)*. Ed. by Andrea Bartolini, Kristian F. D. Rietveld, Catherine D. Schuman, and Jose Moreira. Bologna, Italy: ACM, May 2023, pp. 379–380. DOI: [10.1145/3587135.3591431](https://doi.org/10.1145/3587135.3591431).
- [Sch05] Martin Schoeberl. “JOP: A Java Optimized Processor for Embedded Real-Time Systems”. PhD thesis. Vienna University of Technology, 2005. URL: <https://www.jopdesign.com/thesis/thesis.pdf> (visited on 07/07/2025).
- [Sch+23b] David Schrammel, Moritz Waser, Lukas Lamster, Martin Unterguggenberger, and Stefan Mangard. “SPEAR-V: Secure and Practical Enclave Architecture for RISC-V”. In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)*. Ed. by Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik. Melbourne, VIC, Australia: ACM, July 2023, pp. 457–468. DOI: [10.1145/3579856.3595784](https://doi.org/10.1145/3579856.3595784).

- [SGD08] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. “Practical Setup Time Violation Attacks on AES”. In: *Proceedings of the 7th European Dependable Computing Conference (EDCC)*. Kaunas, Lithuania: IEEE Computer Society, May 2008, pp. 91–96. doi: [10.1109/EDCC-7.2008.11](https://doi.org/10.1109/EDCC-7.2008.11).
- [Sil24] Mário da Silva Araújo. “Security analysis of open-source RISC-V processors”. Master’s thesis. Gardanne, France: École des Mines de Saint-Étienne, Sept. 2024.
- [Sim20] Boris Simunovic. “Security Analysis of the ISO-7816 Stack”. Master’s thesis. Valence, France: ESISAR, Sept. 2020.
- [Sir99] Emin Gü Sire. “Testing Java Virtual Machines: An Experience Report on Automatically Testing Java Virtual Machines”. In: *Proceedings of the International Conference on Software Testing And Review*. San Jose, California, Nov. 1999.
- [Slu+23] pcy Sluys, Lennert Wouters, Benedikt Gierlichs, and Ingrid Verbauwhede. “An In-Depth Security Evaluation of the Nintendo DSi Gaming Console”. In: *Proceedings of the 22nd International Smart Card Research and Advanced Applications (CARDIS) Conference, Revised Selected Papers*. Ed. by Shivam Bhasin and Thomas Roche. Vol. 14530. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Nov. 2023, pp. 23–42. doi: [10.1007/978-3-031-54409-5\\_2](https://doi.org/10.1007/978-3-031-54409-5_2).
- [SCW19] Jinho Song, Chaeho Cho, and Yoojae Won. “Analysis of operating system identification via fingerprinting and machine learning”. In: *Computers & Electrical Engineering* 78 (2019), pp. 1–10. doi: [10.1016/j.compeleceng.2019.06.012](https://doi.org/10.1016/j.compeleceng.2019.06.012).
- [Sve+22] Petr Svenda, Rudolf Kvasnovský, Imrich Nagy, and Antonín Dufka. “JCAIlgTest: Robust Identification Metadata for Certified Smartcards”. In: *Proceedings of the 19th International Conference on Security and Cryptography (SECRYPT)*. Ed. by Sabrina De Capitani di Vimercati and Pierangela Samarati. Lisbon, Portugal: SCITEPRESS, July 2022, pp. 597–604. doi: [10.5220/0011294000003283](https://doi.org/10.5220/0011294000003283).
- [TSS17] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. “CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management”. In: *Proceedings of the 26th USENIX Security Symposium*. Ed. by Engin Kirda and Thomas Ristenpart. Vancouver, BC, Canada: USENIX Association, Aug. 2017, pp. 1057–1074.
- [Tho22] Romain Thomas. “DroidGuard: A Deep Dive into SafetyNet”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)* (June 2022). URL: [https://www.sstic.org/2022/presentation/droidguard\\_a\\_deep\\_dive\\_into\\_safetynet/](https://www.sstic.org/2022/presentation/droidguard_a_deep_dive_into_safetynet/) (visited on 07/07/2025).
- [TM17] Niek Timmers and Cristofaro Mune. “Escalating Privileges in Linux Using Voltage Fault Injection”. In: *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Taipei, Taiwan: IEEE Computer Society, Sept. 2017, pp. 1–8. doi: [10.1109/FDTC.2017.16](https://doi.org/10.1109/FDTC.2017.16).
- [TSW16] Niek Timmers, Albert Spruyt, and Marc Witteman. “Controlling PC on ARM Using Fault Injection”. In: *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. Santa Barbara, California, USA: IEEE Computer Society, Aug. 2016, pp. 25–35. ISBN: 978-1-5090-1108-7. doi: [10.1109/FDTC.2016.18](https://doi.org/10.1109/FDTC.2016.18).
- [Tol24] Simon Tollec. “Formal verification of processor microarchitecture to analyze system security against fault attacks”. PhD thesis. Saclay, France: Université Paris-Saclay, Nov. 2024.

- [Tol+23] Simon Tollec, Mihail Asavoae, Damien Couroussé, Karine Heydemann, and Mathieu Jan. “ $\mu$ ARCHIFI: Formal Modeling and Verification Strategies for Microarchitectural Fault Injections”. In: *Proceedings of the Formal Methods in Computer-Aided Design (FMCAD)*. Ed. by Alexander Nadel and Kristin Yvonne Rozier. Ames, IA, USA: IEEE, Oct. 2023, pp. 101–109. doi: [10.34727/2023/ISBN.978-3-85448-060-0\\_18](https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_18).
- [Tol+24] Simon Tollec, Vedad Hadzic, Pascal Nasahl, Mihail Asavoae, Roderick Bloem, Damien Couroussé, Karine Heydemann, Mathieu Jan, and Stefan Mangard. “Fault-Resistant Partitioning of Secure CPUs for System Co-Verification against Faults”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems 2024.4* (Sept. 2024), pp. 179–204. doi: [10.46586/TCHES.V2024.I4.179-204](https://doi.org/10.46586/TCHES.V2024.I4.179-204).
- [Tou22] Bill Toulas. *New Intel chips won't play Blu-ray disks due to SGX deprecation*. Jan. 2022. URL: <https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/> (visited on 07/07/2025).
- [TB25] Philippe Trébuchet and Guillaume Bouffard. “300 secondes chrono: prise de contrôle d’un infodivertissement automobile à distance”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. June 2025.
- [Tro17] Thomas Troughkine. “Hardware Implementation of a Java Card Virtual Machine”. Master’s thesis. Gardanne, France: École des Mines de Saint-Étienne, Sept. 2017.
- [Tro21] Thomas Troughkine. “System-on-Chip Physical Security Evaluation”. PhD thesis. Grenoble, France: Université Grenoble Alpes, Mar. 2021.
- [TBC19] Thomas Troughkine, Guillaume Bouffard, and Jessy Clédière. “Fault Injection Characterization on Modern CPUs”. In: *Proceedings of the 13th International Conference Information Security Theory and Practice (WISTP)*. Ed. by Maryline Laurent and Thanassis Giannetsos. Vol. 12024. Lecture Notes in Computer Science. Paris, France: Springer, Dec. 2019, pp. 123–138. doi: [10.1007/978-3-030-41702-4\\_8](https://doi.org/10.1007/978-3-030-41702-4_8).
- [TBC21] Thomas Troughkine, Guillaume Bouffard, and Jessy Clédière. “EM Fault Model Characterization on SoCs: From Different Architectures to the Same Fault Model”. In: *Proceedings of the 18th Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. Milan, Italy: IEEE, Sept. 2021, pp. 31–38. doi: [10.1109/FDTC53659.2021.00014](https://doi.org/10.1109/FDTC53659.2021.00014).
- [Tro+21] Thomas Troughkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, and Guillaume Bouffard. “Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models”. In: *Journal of Cryptographic Engineering (JCEN)* (Mar. 2021). doi: [10.1007/s13389-021-00259-6](https://doi.org/10.1007/s13389-021-00259-6).
- [Tur96] Jim Turley. “Sun reveals first Java processor core”. In: *Microprocessor Report* 10.14 (1996), pp. 28–31.
- [Vas+17] Aurélien Vasselle, Hugues Thiebauld, Quentin Maouhoub, Adèle Morisset, and Sébastien Ermeneux. “Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot”. In: *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. aipei, Taiwan: IEEE Computer Society, Sept. 2017, pp. 41–48. doi: [10.1109/FDTC.2017.18](https://doi.org/10.1109/FDTC.2017.18).
- [Vas+20] Aurélien Vasselle, Hugues Thiebauld, Quentin Maouhoub, Adèle Morisset, and Sébastien Ermeneux. “Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot-Extended Version”. In: *IEEE Transactions on Computers* 69.10 (Oct. 2020), pp. 1449–1459. doi: [10.1109/TC.2018.2860010](https://doi.org/10.1109/TC.2018.2860010).
- [Vin15] Guillaume Vinet. “Compromission de carte à puce via la couche protocolaire ISO 7816-3”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2015.

- [Wer+23] Christian Werling, Niclas Kühnapfel, Hans Niklas Jacob, and Oleg Drokin. “Back in the Driver’s Seat: Recovering Critical Data from Tesla Autopilot Using Voltage Glitching”. In: *Proceedings of the 37th Chaos Communication Congress* (Dec. 2023). URL: [https://media.ccc.de/v/37c3-12144-back\\_in\\_the\\_driver\\_s\\_seat\\_recovering\\_critical\\_data\\_from\\_tesla\\_autopilot\\_using\\_voltage\\_glitching](https://media.ccc.de/v/37c3-12144-back_in_the_driver_s_seat_recovering_critical_data_from_tesla_autopilot_using_voltage_glitching).
- [Wer+19] Mario Werner, Robert Schilling, Thomas Unterluggauer, and Stefan Mangard. “Protecting RISC-V Processors against Physical Attacks”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Ed. by Jürgen Teich and Franco Fummi. Florence, Italy: IEEE, Mar. 2019, pp. 1136–1141. doi: [10.23919/DATE.2019.8714811](https://doi.org/10.23919/DATE.2019.8714811).
- [Wid17] DRM Widevine. “Architecture Overview”. In: *Google-Confidential*. Google Inc (2017).
- [Wik24a] Wikipedia. *Commercial off-the-shelf*. May 2024. URL: [https://en.wikipedia.org/wiki/Commercial\\_off-the-shelf](https://en.wikipedia.org/wiki/Commercial_off-the-shelf) (visited on 07/07/2025).
- [Wik24b] Wikipedia. *ISO/IEC 7816*. Aug. 2024. URL: [https://en.wikipedia.org/wiki/ISO/IEC\\_7816](https://en.wikipedia.org/wiki/ISO/IEC_7816) (visited on 07/07/2025).
- [Wik24c] Wikipedia. *Microarchitecture*. May 2024. URL: <https://en.wikipedia.org/wiki/Microarchitecture> (visited on 07/07/2025).
- [Wik24d] Wikipedia. *Protection Profile*. June 2024. URL: [https://en.wikipedia.org/wiki/Protection\\_Profile](https://en.wikipedia.org/wiki/Protection_Profile) (visited on 07/07/2025).
- [Woo+14] Jonathan Woodruff, Robert N. M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert M. Norton, and Michael Roe. “The CHERI capability model: Revisiting RISC in an age of risk”. In: *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. Minneapolis, MN, USA: IEEE Computer Society, June 2014, pp. 457–468. doi: [10.1109/ISCA.2014.6853201](https://doi.org/10.1109/ISCA.2014.6853201).
- [Yah23] Taro Yahagi. *Anti-cheat and Anti-Piracy Measures in PC Games Recommendations for In-House Production*. Oct. 2023. URL: <https://www.capcom-games.com/coc/2023/en/session/04/> (visited on 07/07/2025).
- [Yan+15] Lin Yan, Yao Guo, Xiangqun Chen, and Hong Mei. “A Study on Power Side Channels on Mobile Devices”. In: *Proceedings of the 7th Asia-Pacific Symposium on Internetware*. Ed. by Hong Mei, Jian Lü, Xiaoxing Ma, Qianxiang Wang, Gang Yin, and Xiaofei Liao. Wuhan, China: ACM, Nov. 2015, pp. 30–38. doi: [10.1145/2875913.2875934](https://doi.org/10.1145/2875913.2875934).
- [YSW18] Bilgiday Yuce, Patrick Schaumont, and Marc Witteman. “Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation”. In: *Journal of Hardware and Systems Security* 2.2 (2018), pp. 111–130. doi: [10.1007/s41635-018-0038-1](https://doi.org/10.1007/s41635-018-0038-1).
- [ZDS23] Lukás Zaoral, Antonin Dufka, and Petr Svenda. “The Adoption Rate of JavaCard Features by Certified Products and Open-Source Projects”. In: *Proceedings of the 22nd International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Shivam Bhasin and Thomas Roche. Vol. 14530. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Nov. 2023, pp. 169–189. doi: [10.1007/978-3-031-54409-5\\_9](https://doi.org/10.1007/978-3-031-54409-5_9).
- [Zel+08] Nickolai Zeldovich, Hari Kannan, Michael Dalton, and Christos Kozyrakis. “Hardware Enforcement of Application Security Policies Using Tagged Memory”. In: *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Ed. by Richard Draves and Robbert van Renesse. San Diego, California, USA: USENIX Association, Dec. 2008, pp. 225–240. URL: [https://www.usenix.org/events/osdi08/tech/full\\_papers/zeldovich/zeldovich.pdf](https://www.usenix.org/events/osdi08/tech/full_papers/zeldovich/zeldovich.pdf) (visited on 07/07/2025).

- 
- [Zha+18] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon D. Koutsoukos, and Gabor Karsai. “Triggering Rowhammer Hardware Faults on ARM: A Revisit”. In: *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security (ASHES@CCS)*. Ed. by Chip-Hong Chang, Ulrich Rührmair, Daniel E. Holcomb, and Jorge Guajardo. Toronto, ON, Canada: ACM, Oct. 2018, pp. 24–33. doi: [10.1145/3266444.3266454](https://doi.org/10.1145/3266444.3266454).
- [Zho+22] Yadi Zhong, Ayush Jain, M. Tanjidur Rahman, Navid Asadizanjani, Jiafeng Xie, and Ujjwal Guin. “AFIA: ATPG-Guided Fault Injection Attack on Secure Logic Locking”. In: *Journal of Electronic Testing* 38.5 (Nov. 2022), pp. 527–546. doi: [10.1007/S10836-022-06028-5](https://doi.org/10.1007/S10836-022-06028-5).



# Appendix A

## Curriculum vitæ

### A.1 Administrative Information

#### Personal Information

*Name:* Guillaume Bouffard  
*Marital Status:* Married  
*Nationality:* French  
*Date of Birth:* May 1, 1987

### A.2 Professional Experience and Degrees

#### A.2.1 Professional Experience

Since Nov. 2014    **Expert in Embedded Software Security**  
ANSSI, Paris

- Study of embedded systems security.
- Providing support to ANSSI beneficiaries.
- Technical support for security evaluations ([Common Criteria](#), [CPSN](#)).
- Since 2023: [Hardware and Software Architectures Lab \(LAM\)](#)
- From 2014 to 2022: [Component Security Lab \(LSC\)](#)

2019 — 2024    **Associate Researcher**  
[Ecole Normale Supérieure \(ENS\)](#), [Computer Science Department](#), [Security Team](#),  
Paris

## A.2.2 Academic Degrees

2011 — 2014

**PhD in Computer Science**, Doctoral School S2IM, Limoges

Thesis conducted at the [Xlim](#) research institute under the supervision of [Pr. Jean-Louis Lanet](#)

Title: **A Generic Approach to Protect Java Card Smartcards Against Software Attacks**

- [Erik Poll](#), President, Associate Professor, Radboud University, Netherlands
- [David Naccache](#), Reviewer, Professor, [ENS Paris](#): report in [Appendix A.7.1](#)
- [Peter Ryan](#), Reviewer, Professor, University of Luxembourg: report in [Appendix A.7.2](#)
- [Jean-Louis Lanet](#), Professor, INRIA
- [Emmanuel Prouff](#), Examiner, HDR, [ANSSI/Sorbonne University](#)
- [Eric Vétillard](#), Examiner, Oracle

The official defense report can be found in [Appendix A.7.3](#). I received the **PhD Award of the CNRS “Objets intelligents sécurisés et Internet des objets” initiative** in 2015.

Smartcards are used in a variety of applications (payment, telephony, etc.). To strengthen their security, Java technology was integrated into operating systems in the 1990s, becoming the main application platform. Since these operating systems are frequent attack targets, continuous adaptation of countermeasures is required. We use Fault Tree Analysis, adapted to Java Card technology constraints, to ensure data and code integrity and confidentiality. Focusing on code integrity, we discovered new vulnerabilities and proposed countermeasures that have a significant impact on the security evaluations carried out by [ITSEFs](#) and contributed to improving the security of products developed by industrial stakeholders.

2008 — 2010

**Master’s Degree in Information Security, Mathematics, and Computer Science (Information Security track – [CRYPTIS](#))**

Faculty of Science and Technology, University of Limoges

Year-end project: **Analysis and transformation of binary code**. Protection of sensitive parts of binary code without access to the source code, carried out during my internship at [Technicolor](#). Sensitive instructions were replaced by a communication mechanism with a USB dongle, which executed the protected instructions and allowed the application to run only if the key was present.

The [CRYPTIS](#) Master’s Program trains specialists in the security of distributed information systems, as well as in the development of secure software and hardware.

## A.3 Teaching

Year	Course	Hours/session	Total
From 2018 to 2024 (6 years)	<i>Hardware to Software Security</i> module at <a href="#">CY-Tech</a> , University of Cergy	28h/year	168h
Since 2017 (7 years)	<i>Cybersecurity of Embedded Systems</i> training program at École Polytechnique Executive Education - Lecture on <i>Embedded Software Security</i>	3.5h/session	60h
From 2015 to 2018 (3 years)	<i>Java Card and Global Platform</i> module at <a href="#">ENSICAEN</a> - Lecture on <i>Java Card Platform Security</i>	7h/year	28h
Since 2015 (10 years)	Teaching on behalf of the <a href="#">CFSSI</a> : - Course 10: <i>Cryptography</i> - Lecture on <i>Cryptography in Secure Components</i> - Course 23: <i>Security of Components</i> - Lecture on <i>Java Card Security</i> - Course 25: <i>Embedded Systems Security</i> - Lecture on <i>Trusted Boot Chain</i>	(Since 2025) 1.5h/session   1.5h (Until 2023) 3.5h/session   31.5h (Since 2025) 3.5h/session   3.5h	
<b>Total:</b>			<b>289h</b>

### A.3.1 Teaching Responsibilities

From 2018 to 2024, I was in charge of the “Hardware to Software Security” course unit at [CY-Tech](#), University of Cergy, for last-year students. This course, which hosted around sixty students split into two groups, covered how the different layers of an information system work and the associated security issues.

### A.3.2 Expert Committee for the Recruitment of Associate Professors

2019: for the University of Cergy-Pontoise, position in CNU 27.

### A.3.3 Participation in MOOCs

- Participation in the MOOC “[Hardware Security: the Hardware Attacks](#)” as an expert on hardware attacks for [ANSSI](#). The video is available on YouTube: [▶ Interview ANSSI](#) (in French).

## A.4 Research Activities at ANSSI

The objective of the [ANSSI](#) laboratories is to provide high-level expertise to all of the agency’s beneficiaries. My duties revolve around three main and complementary areas:

### 1. Expertise:

- Technical support for the operational teams at ANSSI.
- Technical support for the National Certification Center, an entity under ANSSI responsible for overseeing and accrediting ITSEFs, as well as certifying products evaluated in France.
- Participation in the technical working groups of GlobalPlatform and JHAS, contributing to the security evaluation methodologies for products.
- Technical collaboration with our foreign counterparts (such as BSI, NLNCSA) in the context of security evaluations, the development of shared specifications, joint work, and the enforcement of European regulations.
- Technical support for European projects (e.g., the European Digital Identity project) and national projects (e.g., France Identité Numérique).
- Technical support for national and international audits of ITSEFs carried out within the framework of the Common Criteria.

### 2. Research and Development (approximately 33% of my working time):

- Studying the security of embedded software, anticipating risks associated with software and hardware attacks, and designing original countermeasures. This work is informed by my expertise activities and, in turn, continuously enhances them.
- Supervising Ph.D. candidates and research interns.

### 3. Training:

- Designing and delivering training sessions for the CFSSI, a training center under ANSSI. These sessions, which are also offered to other external audiences, directly leverage my expertise and research activities, ensuring content that is relevant, up-to-date, and illustrated with concrete examples.
- Participating in events that foster the dissemination of cybersecurity knowledge.

## A.4.1 My Research Activities

My research focuses on the security of embedded software within the CoT. The CoT relies on a secure architecture where each component, from the RoT to the TEE, ensures the reliability and integrity of higher-level layers, while isolating less secure environments such as Android and other REEs, thus protecting the overall system. An overview of my research activities is provided in Figure 1.3, and detailed throughout this manuscript in subsequent chapters, notably Chapters 2 to 4.

**Question 1: How are the local and platform security functions, provided by hardware RoTs, designed and leveraged to reinforce security?** My doctoral thesis [Bou14] focused on the security of embedded software implementations in SEs. Subsequently, my research extended to the study of hardware RoTs, examining their integration into a CoT from both the developer’s and user’s perspective. The goal is to anticipate risks and ensure effective use of these critical security elements.

**Question 2: How can a high level of security be guaranteed in TEEs, which offer a high-performance environment for business application security functions?** The integration of TEEs into application processors, such as Arm TrustZone [Arm17], has strengthened the security of sensitive applications by isolating their critical operations in a secure enclave. The certification of the TEE protection profile by ANSSI [Glo20b] was a major step forward, although hardware attacks remain insufficiently addressed. Yet these attacks, initially targeting hardware RoTs [Vas+20; TSW16], now exploit the broader attack surface of application processors, making their protection more complex.

**Question 3: How can sensitive applications operate in the REE, an environment lacking security features?** In many cases, access to TEEs and hardware RoTs is restricted due to platform vendor agreements. Sensitive applications thus have to run in the REE, where they are exposed to white-box attacks [Cho+02]. Numerous software solutions protected by DRM mechanisms [Yah23; App16] or proprietary solutions, notably for streaming or video games [Heu24], are deployed in this environment. Even some critical payment applications are installed here, significantly increasing risks [Pay20]. The scientific community strives to identify and analyze these vulnerabilities [Bar+22]. It is therefore essential to evaluate the feasibility of using obfuscated applications as an alternative to RoTs in the REE, in order to guide their adoption when the use of a RoT or TEE is not possible.

## A.4.2 Past and Ongoing Scientific Collaborations

### Participation in Funded Projects<sup>1</sup>:

- 2025 — 2029 PTCC FORWARD  
(48 months) *Goal:* ensure the resilience of hardware countermeasures by using formal verification against physical attacks.  
*Partners:* CEA (project leader), INRIA, Mines Saint-Étienne, Sorbonne Université, ANSSI, Thales DIS, and Safran.  
**Contributions:** selecting secure architectures, overseeing fault experiments, and formalizing conclusions and perspectives.  
*Total grant:* 1 226 500 euros / *total budget:* 2 213 450 euros
- 2023 — 2029 PEPR ARSENE  
(60 months) *Goal:* accelerate the research, development, and demonstration of sovereign and industrializable security solutions, using ASIC and FPGA demonstrators to test and showcase research findings.  
*Partners:* CEA (project leader), INRIA, CNRS, and ANSSI.  
**Participation in Work Package 1: Secure RISC-V:** studying and validating protections against hardware attacks for a 32-bit core.  
*Total grant:* 7 900 000 euros / *total budget:* 12 440 000 euros

### Participation in Funded Projects as a Member of the Scientific Committee:

- 2021 — 2025 ANR TrustGW  
(42 months) *Goal:* Develop a heterogeneous software-hardware architecture for a gateway that can be dynamically reconfigured and trusted.  
*Partners:* Lab-STICC (project leader), IRISA, IETR.  
*Total grant:* 465 752 euros.
- 2021 — 2025 ANR Secure-V  
(42 months) *Goal:* Design a secure, open, high-performance processor based on the RISC-V architecture.  
*Partners:* IETR (project leader) (University of Nantes), LS2N (University of Nantes, Centrale Nantes), IMS (University of Bordeaux), TRT (Thales), INVIA (Thales).  
*Total grant:* 731 831 euros

1. ANSSI co-funds projects via the SGDSN, which restricts our involvement to in-house funding. This limits our direct contributions but allows us to follow projects through a scientific committee and to develop parallel collaborations.

- 2017 — 2020 **FUI SecurIoT-2**  
 (36 months) *Goal:* Develop a secure microcontroller for the next generation of IoT devices and connected objects.  
*Partners:* Tiempo Secure (project leader), Archos, Alpwis, Sensing Labs, Trusted Objects, INRIA Rhone Alpes, University of Grenoble Alpes.  
*Total budget:* 5 410 000 euros
- 2015 — 2018 **FUI TEEVA**  
 (36 months) *Goal:* Analyze the security level of TEE technologies and white-box cryptography against software and hardware attacks on smartphone processors.  
*Partners:* Gemalto (project leader), Trustonic, Phossec, LIRMM/University of Montpellier, ENSMSE Gardanne, LHC/University Saint Etienne.  
*Total budget:* 3 161 000 euros

### Collaborative Works

My research activities would not have been possible without the collaborations developed over the course of my work. Below is a list, organized by institution, of external researchers with whom I have had the opportunity to collaborate:

#### 1. ANSSI

- **Luc Bonnafoux:** collaboration on hardware implementation security during the supervision of **Angie-Sofia Bikou** and **Guillaume P.** internships.
- **Patrick Haddad:** collaboration on the design of a hardware JCVM on FPGA during the supervision of **Thomas Trouchkine**'s internship.
- **Valentin Houchouas** and **José Lopes Esteves:** collaboration on the reproducibility of EMFI experiments, with the goal of providing sufficient information in scientific publications to allow third parties to reproduce the setups and results.
- **Louiza Khati:** collaboration on the security of disk encryption during the supervision of **Yanis Belkheyar**'s internship.
- **Philippe Trébuchet:** collaboration on analyzing the cybersecurity of connected vehicles.

#### 2. Arm

- **Arnaud de Grandmaison:** collaboration on software security against hardware attacks by supervising **Ever Atilano Rosales**'s internship.

#### 3. CEA/Leti

- **Jessy Clédière:** collaboration on characterizing the effects of fault injection attacks on application SoCs. Supervised the Ph.D. of **Thomas Trouchkine**.

#### 4. CEA/List

- **Damien Couroussé** and **Mathieu Jan:** collaboration on analyzing implementation security against hardware attacks through formal approaches, by supervising the Ph.D. of **Jonah Alle Monne** and the internship of **Mário da Silva Araújo**.

#### 5. DGA-MI

- **Rachid Dafali:** studying the effects of fault injection attacks on micro-architecture. Participation in co-supervising the Ph.D. of **Amélie Marotta**.

## 6. ENS

- **David Naccache**: studying software security in an uncontrolled execution environment. Supervisor of the Ph.D. of [Vincent Giraud](#).

## 7. IETR

- **Jean-Christophe Prévotet**: studying the security of [TEEs](#) against hardware attacks. Co-supervisor of the Ph.D. of [Gwenn Le Gonidec](#).

## 8. INRIA

- **Ronan Lashermes** (LHS) and **Olivier Sentieys** (TARAN): studying the effects of fault injection attacks on micro-architecture. Co-supervisors of the Ph.D. of [Amélie Marotta](#).

## 9. Sorbonne Université/LIP6

- **Karine Heydemann**: collaboration on software security against hardware attacks. We worked together around the Ph.D. of [Thomas Troughkine](#) and co-supervised [Ever Atilano Rosales](#)'s internship (Arm).

## 10. Université Bretagne-Sud/Lab-STICC

- **Maria Méndez Real**: studying the security of [TEEs](#) against hardware attacks. Co-supervisor of the Ph.D. of [Gwenn Le Gonidec](#).

## A.5 Scientific Responsibilities

### A.5.1 Ph.D. Supervision

#### Co-supervised and Defended Theses

- 2021 – 2025 [Amélie Marotta](#), *Effects of synchronous clock glitch on the security of integrated circuits*  
Co-supervised with [Olivier Sentieys](#) (INRIA, director), [Ronan Lashermes](#) (INRIA, co-director) and [Rachid Dafali](#) (DGA-MI)  
Supervision rate: 25%. This thesis took place at INRIA in Rennes.  
This thesis took place at INRIA in Rennes.  
Doctoral School: Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique (ED 601 – MATISSE)  
Specialty: Computer Science  
Publication: [[Mar+24](#)]
- 2020 – 2024 [Vincent Giraud](#), *Security of Applications on Uncontrolled Systems: Risk Analysis, Protections, Challenges, and the Value of Trust in Off-the-shelf IT Products*  
Co-supervised with [David Naccache](#) ([ENS](#), director)  
Supervision rate: 75%. This thesis took place at Ingenico in Paris.  
Doctoral School: Sciences Mathématiques de Paris Centre (ED 386 – SPMC)  
Specialty: Computer Science  
Publications: [[GB23](#); [GN23a](#); [GN23b](#)]  
Patents: [[Gir23a](#); [Gir23b](#); [Gir23c](#)]

- 2017 – 2021 Thomas Troughkine, *Physical Security Evaluation of SoCs*  
 Co-supervised with Jessy Clédière (CEA/Leti, director)  
 Supervision rate: 75%. This thesis took place at ANSSI in Paris.  
 Doctoral School: Électronique, Électrotechnique, Automatique, Traitement du Signal (ED 220 – EEATS)  
 Specialty: Nanoelectronics and Nanotechnologies  
 Publications: [TBC21; Tro+21; TBC19]

## Ongoing Theses

- Since 2024 Jonah Alle Monne, *Formalization and Analysis of Countermeasures against Fault Injection Attacks on Open-source Processors*.  
 Co-supervised with Damien Couroussé (CEA/List, director) and Mathieu Jan (CEA/List, co-director)  
 Supervision rate: 33%. This thesis is taking place at CEA/List in Grenoble.  
 Doctoral School: Mathématiques, Sciences et Technologies de l'Information, Informatique (ED 217 – MSTII)  
 Specialty: Computer Science
- Since 2023 Gwenn Le Gonidec, *Securing RISC-V System-on-Chip against Energy-based Attacks*  
 Co-supervised with Maria Méndez Real (Lab-STICC, director) and Jean-Christophe Prévotet (INSA/IETR, co-director)  
 Supervision rate: 35%. This thesis is taking place at Lab-STICC in Lorient.  
 Doctoral School: Mathématiques & Sciences et Technologies de l'Information et de la Communication en Bretagne Océane (ED 644 – MathSTIC Bretagne Océane)  
 Specialty: Electronics  
 Publication: [Gon+25]

## A.5.2 Apprenticeship Supervision

- 2017 to 2020 Boris Simunovic: *Security Analysis of the ISO-7816 Protocol Stack*.

## A.5.3 Internship Supervision

The internships I supervise, each lasting 6 months, are research internships at the Master 2 level.

- 2025 Guillaume P., *Implementation of a security architecture on an Arm COTS SoC*.  
 Co-supervised with Luc Bonnafox (ANSSI/LAM).
- 2024 Angie-Sofia Bikou, *Analysis of a Secure Component Architecture*.  
 Co-supervision with Luc Bonnafox (ANSSI/LAM).
- 2024 Mário da Silva Araújo, *Security Analysis of RISC-V Processors in the Age of Open Source*.  
 Co-supervision with Damien Couroussé (CEA/List), Mathieu Jan (CEA/List) and Simon Tollec (CEA/List).
- 2022 Louisa Malki-Haegel, *Measurement of Embedded Software Footprint*.
- 2021 Ever Atilano Rosales, *Securing Secure Boot Against Fault Injection Attacks*.  
 Co-supervision with Arnaud de Grandmaison (Arm) and Karine Heydemann (Sorbonne Université/LIP6). This work was presented at the Linaro 2021 conference<sup>2</sup>.
- 2020 Yanis Belkheyar, *Authenticated Disk Encryption*.

- Co-supervision with Louiza Khati ([ANSSI/Cryptography Lab](#)).
- 2019 Vincent Giraud, *Secure Implementation of GlobalPlatform for Java Card Platforms*.
- 2017 Thomas Troughkine, *Development of a Native Java Card Processor on a Virtex 5 FPGA*.  
Co-supervision with Patrick Haddad ([ANSSI/LSC](#)).
- 2016 Léo Gaspard, *Implementation of a Secure OS for Java Card Platforms*.  
Part of this work was published at SSTIC 2018 [[BG18](#)].

## A.5.4 Scientific Involvement

### Conference Organization

- Since 2023 Member of the scientific committee of the [European Cyber Week](#) conference, participating in the [BITFLIP by DGA](#) workshop, organized by the DGA Maîtrise de l'Information. This event highlights recent advances in failure analysis for digital systems, covering both radiation hardening and protection against intentional disruption attacks. This conference is held every 2 years, and we welcome around 80 on-site participants
- Since 2018 Co-organization of the [Thematic Days on Fault Injection Attacks](#) (JAIF), gathering annually the French research community working on fault analysis of modern system. These workshops aim to consolidate knowledge and support comprehensive research, gathering around 130 participants on-site, and a similar number attending remotely. I organized the 2020 and 2021 editions at [ENS Paris](#).

### Session Chair in Conferences

- 2021 Session chair for *short papers* at [FDTC 2021](#).
- 2016 Session chair for *Java Card* at [CARDIS 2016](#).

### Journal Article Reviewer

- 2024 [ACM Transactions on Embedded Computing Systems](#).
- 2023 [International Journal of Information and Computer Security](#).
- 2020 [ACM Transactions on Privacy and Security](#).
- 2020 [ACM Digital Threats: Research and Practice](#): Special issue on Threats of Hardware Security.

### Program Committee Membership and Reviewing

- 2026 [CASCADE](#): Workshop on Embedded System Security.
- Since 2022 [FDTC](#): Workshop on Fault Diagnosis and Tolerance in Cryptography.
- Since 2018 [SecITC](#): International Conference on Information Technology and Communication Security
- 2024 [DRIN](#): Workshop on DevSecOps in Resource-Constrained IoT Networks

---

2. Ever Atilano, Arnaud Grandmaison, "Assessing the effectiveness of MCUBoot protections against fault injection attacks". In: *Linaro Connect Virtual Connect Fall* (Sept. 2021). URL: <https://resources.linaro.org/en/resource/ibFLwRzhpZjBfvY5jhPypJ> (visited on 07/07/2025).

- 2021 [SILM](#): Workshop on the Security of Software/Hardware Interfaces.
- 2020 [TrustCom](#): IEEE International Conference on Trust, Security and Privacy in Computing and Communications
- 2015 [CARDIS](#): Smart Card Research and Advanced Application Conference

### External Conference Reviewer

- 2025 [ICCAD](#): The 2025 International Conference on Computer-Aided Design: Track 1.8 Architecture and Systems for Security
- 2022 [CARDIS](#): Smart Card Research and Advanced Application Conference
- 2020 [CHES](#): Conference on Cryptographic Hardware and Embedded Systems
- 2019 [CARDIS](#): Smart Card Research and Advanced Application Conference
- 2017 [CHES](#): Conference on Cryptographic Hardware and Embedded Systems
- 2017 [C2SI](#): Codes, Cryptology and Information Security
- 2015 [PROOFS](#): International Workshop on Security Proofs for Embedded Systems

### Expert Assessments

I served as an expert in 2020 and 2021 for the French National Research Agency ([ANR](#)), evaluating projects under the “*CE39 – Global Security, Cybersecurity*” call for proposals.

### Other Activities

- Since 2019 Participation in the international competition [CSAW](#) as a judge for the *Apply Research Competition* challenge.

## A.5.5 Participation in Thesis Defense Committees as Examiner

- 2024 Antoine Gicquel, *Vulnerability Analysis of Binary Programs Under Multiple Precise Faults: Metrics and Countermeasures*. Co-supervisors: Damien Hardy, Karine Heydemann, and Erven Rohou.
- 2024 Simon Tollec, *Formal Micro-Architecture Verification for Fault Injection Effects and Countermeasure Robustness*. Co-supervisors: Mihail Asavoaie, Damien Couroussé, Karine Heydemann, and Mathieu Jan.
- 2023 Soline Ducouso, *Transitioning from Safety to Security in Code Analysis: the Attacker Model*. Co-supervisors: Sébastien Bardin and Marie-Laure Potet.
- 2022 Vincent Werner, *Optimizing the Identification and Exploitation of Fault Injection Vulnerabilities in Microcontrollers*. Co-supervisors: Laurent Maingault and Marie-Laure Potet.
- 2021 Youssef Inedjaren, *Contribution to Intelligent Transport Systems: Securing Communications in Vehicular Ad Hoc Networks*. Co-supervisors: Jean-Pierre Barbot, Mohamed Maachaoui, and Besma Zeddini.
- 2019 Sebanjila Kevin Bukasa, *Vulnerability Analysis of Embedded Systems Against Physical Attacks*. Co-supervisors: Jean-Louis Lanet and Ronan Lashermes.
- 2018 Damien Marion, *Multidimensionality of Models and Data in Side-Channel Analysis*. Co-supervisors: Adrien Facon and Sylvain Guilley.
- 2018 Abdelhak Mesbah, *Reconstructing Execution Traces from Fragments by Using Constraint Solvers*. Co-supervisors: Jean-Louis Lanet and Mohamed Mezghiche.

- 2016 Tiana Razafindralambo, *Security of Embedded Microcontrollers: from Smart Cards to Mobile Devices*. Co-supervisors: Christophe Clavier and Jean-Louis Lanet.
- 2016 Louis Dureuil, *Code Analysis and Evaluation Processes of Secure Components Against Fault Injection*. Co-supervisors: Philippe de Choudens and Marie-Laure Potet.

### A.5.6 Participation in Mid-term Ph.D. Evaluations

- Mathieu Escouteloup (2019 & 2020)
- Léopold Ouairy (2018 & 2019)

### A.5.7 Scientific Publications

This section lists the publications I have co-authored during and after my doctoral thesis [Bou14]. Table A.14 summarizes and enumerates a selection of my scientific publications.

Type of Publication	Number and References	
	After my Ph.D. (after 2014)	During my Ph.D. (before 2014)
Book Chapters	–	2 [BB13; BL12]
Journal Articles	3 [Gon+25; Tro+21; Idr+17]	4 [BL15; BL14b; BTL14; Dub+13]
International Conferences	6 [Mar+24; GB23; DB21; TBC21; TBC19; LB15]	10 [Bou+14; Lan+14; Bou+13a; BTL13a; BTL13b; Dub+12; RBL12; Raz+12; BIL11; Bou+11]
National Conferences	3 [TB25; BG18; LB16]	4 [BL14a; Bou+13b; Ham+12; Nou+09]

**Table A.14** – Listing of published works. A digital copy of each article is available on HAL and on my personal website.

The full list of my publications can be found in section A.6. Below, I highlight seven key publications that illustrate the quality and breadth of my research in computer security.

- [Gon+25] Gwenn Le Gonidec, **Guillaume Bouffard**, Jean-Christophe Prévotet, Maria Méndez Real, “Do Not Trust Power Management: A Survey on Internal Energy-based Attacks Circumventing Trusted Execution Environments Security Properties”. In: *ACM Transactions on Embedded Computing Systems* 24.4 (July 2025). ISSN: 1539-9087. DOI: [10.1145/3735556](https://doi.org/10.1145/3735556).
- [TB25] Philippe Trébuchet, **Guillaume Bouffard**, “300 secondes chrono: prise de contrôle d’un infodivertissement automobile à distance”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. June 2025.
- [Mar+24] Amélie Marotta, Ronan Lashermes, **Guillaume Bouffard**, Olivier Sentieys, Rachid Dafali, “Characterizing and Modeling Synchronous Clock-Glitch Fault Injection”. In: *Proceedings of the 15th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*. Ed. by Romain Wacquez and Naofumi Homma. Vol. 14595. Lecture Notes in Computer Science. Gardanne, France: Springer, Apr. 2024, pp. 3–21. DOI: [10.1007/978-3-031-57543-3\\_1](https://doi.org/10.1007/978-3-031-57543-3_1).

- [DB21] Jean Dubreuil, **Guillaume Bouffard**, “PhiAttack - Rewriting the Java Card Class Hierarchy”. In: *Proceedings of the 20th International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Lübeck, Germany: Springer, Nov. 2021, pp. 275–288. doi: [10.1007/978-3-030-97348-3\\_15](https://doi.org/10.1007/978-3-030-97348-3_15).
- [TBC21] Thomas Troughkine, **Guillaume Bouffard**, Jessy Clédière, “EM Fault Model Characterization on SoCs: From Different Architectures to the Same Fault Model”. In: *Proceedings of the 18th Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. Milan, Italy: IEEE, Sept. 2021, pp. 31–38. doi: [10.1109/FDTC53659.2021.00014](https://doi.org/10.1109/FDTC53659.2021.00014).
- [Tro+21] Thomas Troughkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, **Guillaume Bouffard**, “Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models”. In: *Journal of Cryptographic Engineering (JCEN)* (Mar. 2021). doi: [10.1007/s13389-021-00259-6](https://doi.org/10.1007/s13389-021-00259-6).
- [LB15] Julien Lancia, **Guillaume Bouffard**, “Java Card Virtual Machine Compromising from a Bytecode Verified Applet”. In: *Proceedings of the 14th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Vol. 9514. Lecture Notes in Computer Science. Bochum, Germany: Springer, Nov. 2015, pp. 75–88. doi: [10.1007/978-3-319-31271-2\\_5](https://doi.org/10.1007/978-3-319-31271-2_5).

## A.6 Full List of My Publications

### Book Chapters

- [BB113] Guillaume Barbu, **Guillaume Bouffard**, Julien Iguchy-Cartigny, “La Sécurité Logique”. In: *Les Cartes à puce*. Ed. by Samia Bouzefrane and Pierre Paradinas. Hermes Science, 2013. Chap. 6, pp. 171–201. ISBN: 9782746239135.
- [BL12] **Guillaume Bouffard**, Jean-Louis Lanet, “The Next Smart Card Nightmare - Logical Attacks, Combined Attacks, Mutant Applications and Other Funny Things”. In: *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*. Ed. by David Naccache. Vol. 6805. Lecture Notes in Computer Science. Springer, 2012, pp. 405–424. ISBN: 978-3-642-28367-3. doi: [10.1007/978-3-642-28368-0\\_26](https://doi.org/10.1007/978-3-642-28368-0_26).

### Journal Articles

- [Gon+25] Gwenn Le Gonidec, **Guillaume Bouffard**, Jean-Christophe Prévotet, Maria Méndez Real, “Do Not Trust Power Management: A Survey on Internal Energy-based Attacks Circumventing Trusted Execution Environments Security Properties”. In: *ACM Transactions on Embedded Computing Systems* 24.4 (July 2025). ISSN: 1539-9087. doi: [10.1145/3735556](https://doi.org/10.1145/3735556).
- [Tro+21] Thomas Troughkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, **Guillaume Bouffard**, “Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models”. In: *Journal of Cryptographic Engineering (JCEN)* (Mar. 2021). doi: [10.1007/s13389-021-00259-6](https://doi.org/10.1007/s13389-021-00259-6).
- [Idr+17] Noredine El Janati El Idrissi, **Guillaume Bouffard**, Jean-Louis Lanet, Said El Hajji, “Trust can be misplaced”. In: *Journal of Cryptographic Engineering* 7.1 (2017), pp. 21–34. doi: [10.1007/s13389-016-0142-5](https://doi.org/10.1007/s13389-016-0142-5).

- [BL15] **Guillaume Bouffard**, Jean-Louis Lanet, “The ultimate control flow transfer in a Java based smart card”. In: *Computers and Security* 50 (May 2015), pp. 33–46. doi: [10.1016/j.cose.2015.01.004](https://doi.org/10.1016/j.cose.2015.01.004).
- [BL14b] **Guillaume Bouffard**, Jean-Louis Lanet, “Reversing the operating system of a Java based smart card”. In: *Journal of Computer Virology and Hacking Techniques* 10.4 (July 2014), pp. 239–253. doi: [10.1007/s11416-014-0218-7](https://doi.org/10.1007/s11416-014-0218-7).
- [BTL14] **Guillaume Bouffard**, Bhagyalekshmy N. Thampi, Jean-Louis Lanet, “Security automaton to mitigate laser-based fault attacks on smart cards”. In: *International Journal of Trust Management in Computing and Communications (IJTMCC)* 2.2 (Sept. 2014), pp. 185–205. doi: [10.1504/IJTMCC.2014.064158](https://doi.org/10.1504/IJTMCC.2014.064158).
- [Dub+13] Jean Dubreuil, **Guillaume Bouffard**, Bhagyalekshmy N. Thampi, Jean-Louis Lanet, “Mitigating Type Confusion on Java Card”. In: *International Journal of Secure Software Engineering* 4.2 (2013), pp. 19–39. doi: [10.4018/jsse.2013040102](https://doi.org/10.4018/jsse.2013040102).

## International Conference Proceedings

- [Mar+24] Amélie Marotta, Ronan Lashermes, **Guillaume Bouffard**, Olivier Sentieys, Rachid Dafali, “Characterizing and Modeling Synchronous Clock-Glitch Fault Injection”. In: *Proceedings of the 15th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*. Ed. by Romain Wacquez and Naofumi Homma. Vol. 14595. Lecture Notes in Computer Science. Gardanne, France: Springer, Apr. 2024, pp. 3–21. doi: [10.1007/978-3-031-57543-3\\_1](https://doi.org/10.1007/978-3-031-57543-3_1).
- [GB23] Vincent Giraud, **Guillaume Bouffard**, “Faulting original McEliece’s implementations is possible. How to mitigate this risk?” In: *IEEE European Workshops on Symposium on Security and Privacy (EuroS&PW)*. Delft, Netherlands: IEEE, July 2023, pp. 311–319. doi: [10.1109/EuroSPW59978.2023.00039](https://doi.org/10.1109/EuroSPW59978.2023.00039).
- [DB21] Jean Dubreuil, **Guillaume Bouffard**, “PhiAttack - Rewriting the Java Card Class Hierarchy”. In: *Proceedings of the 20th International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Vincent Grosso and Thomas Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Lübeck, Germany: Springer, Nov. 2021, pp. 275–288. doi: [10.1007/978-3-030-97348-3\\_15](https://doi.org/10.1007/978-3-030-97348-3_15).
- [TBC21] Thomas Troughkine, **Guillaume Bouffard**, Jessy Clédière, “EM Fault Model Characterization on SoCs: From Different Architectures to the Same Fault Model”. In: *Proceedings of the 18th Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. Milan, Italy: IEEE, Sept. 2021, pp. 31–38. doi: [10.1109/FDTC53659.2021.00014](https://doi.org/10.1109/FDTC53659.2021.00014).
- [TBC19] Thomas Troughkine, **Guillaume Bouffard**, Jessy Clédière, “Fault Injection Characterization on Modern CPUs”. In: *Proceedings of the 13th International Conference Information Security Theory and Practice (WISTP)*. Ed. by Maryline Laurent and Thanassis Giannetsos. Vol. 12024. Lecture Notes in Computer Science. Paris, France: Springer, Dec. 2019, pp. 123–138. doi: [10.1007/978-3-030-41702-4\\_8](https://doi.org/10.1007/978-3-030-41702-4_8).
- [LB15] Julien Lancia, **Guillaume Bouffard**, “Java Card Virtual Machine Compromising from a Bytecode Verified Applet”. In: *Proceedings of the 14th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Vol. 9514. Lecture Notes in Computer Science. Bochum, Germany: Springer, Nov. 2015, pp. 75–88. doi: [10.1007/978-3-319-31271-2\\_5](https://doi.org/10.1007/978-3-319-31271-2_5).

- [Bou+14] **Guillaume Bouffard**, Michael Lackner, Jean-Louis Lanet, Johannes Loinig, “Heap ... Hop! Heap Is Also Vulnerable”. In: *Proceedings of the 13th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Marc Joye and Amir Moradi. Vol. 8968. Lecture Notes in Computer Science. Paris, France: Springer, Nov. 2014, pp. 18–31. ISBN: 978-3-319-16762-6. DOI: [10.1007/978-3-319-16763-3\\_2](https://doi.org/10.1007/978-3-319-16763-3_2).
- [Lan+14] Jean-Louis Lanet, **Guillaume Bouffard**, Rokia Lamrani, Ranim Chakra, Afef Mestiri, Mohammed Monsif, Abdellatif Fandi, “Memory Forensics of a Java Card Dump”. In: *Proceedings of the 13th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Marc Joye and Amir Moradi. Vol. 8968. Lecture Notes in Computer Science. Paris, France: Springer, Nov. 2014, pp. 3–17. ISBN: 978-3-319-16762-6. DOI: [10.1007/978-3-319-16763-3\\_1](https://doi.org/10.1007/978-3-319-16763-3_1).
- [Bou+13a] **Guillaume Bouffard**, Tom Khefif, Jean-Louis Lanet, Ismael Kane, Sergio Casanova Salvia, “Accessing secure information using export file fraudulence”. In: *Proceedings of the International Conference on Risks and Security of Internet and Systems (CRiSIS)*. Ed. by Bruno Crispo, Ravi S. Sandhu, Nora Cuppens-Boulahia, Mauro Conti, and Jean-Louis Lanet. La Rochelle, France: IEEE Computer Society, Oct. 2013, pp. 1–5. DOI: [10.1109/CRiSIS.2013.6766346](https://doi.org/10.1109/CRiSIS.2013.6766346).
- [BTL13a] **Guillaume Bouffard**, Bhagyalekshmy N. Thampi, Jean-Louis Lanet, “Detecting Laser Fault Injection for Smart Cards Using Security Automata”. In: *Proceedings of the International Symposium on Security in Computing and Communications (SSCC)*. Ed. by Sabu M. Thampi, Pradeep K. Atrey, Chun-I Fan, and Gregorio Martínez Pérez. Vol. 377. Communications in Computer and Information Science. Mysore, India: Springer, Aug. 2013, pp. 18–29. ISBN: 978-3-642-40575-4. DOI: [10.1007/978-3-642-40576-1\\_3](https://doi.org/10.1007/978-3-642-40576-1_3).
- [BTL13b] **Guillaume Bouffard**, Bhagyalekshmy N. Thampi, Jean-Louis Lanet, “Vulnerability Analysis on Smart Cards Using Fault Tree”. In: *Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*. Ed. by Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaâniche. Vol. 8153. Lecture Notes in Computer Science. Toulouse, France: Springer, Sept. 2013, pp. 82–93. ISBN: 978-3-642-40792-5. DOI: [10.1007/978-3-642-40793-2\\_8](https://doi.org/10.1007/978-3-642-40793-2_8).
- [Dub+12] Jean Dubreuil, **Guillaume Bouffard**, Jean-Louis Lanet, Julien Cartigny, “Type Classification against Fault Enabled Mutant in Java Based Smart Card”. In: *Proceedings of the 7th International Conference on Availability, Reliability and Security (ARES)*. Prague, Czech Republic: IEEE Computer Society, Aug. 2012, pp. 551–556. ISBN: 978-1-4673-2244-7. DOI: [10.1109/ARES.2012.24](https://doi.org/10.1109/ARES.2012.24).
- [RBL12] Tiana Razafindralambo, **Guillaume Bouffard**, Jean-Louis Lanet, “A Friendly Framework for Hidding fault enabled virus for Java Based Smartcard”. In: *Proceedings of the 26th Annual Conference on Data and Applications Security and Privacy (DBSec)*. Ed. by Nora Cuppens-Boulahia, Frédéric Cuppens, and Joaquín García-Alfaro. Vol. 7371. Lecture Notes in Computer Science. Paris, France: Springer, July 2012, pp. 122–128. DOI: [10.1007/978-3-642-31540-4\\_10](https://doi.org/10.1007/978-3-642-31540-4_10).
- [Raz+12] Tiana Razafindralambo, **Guillaume Bouffard**, Bhagyalekshmy N. Thampi, Jean-Louis Lanet, “A Dynamic Syntax Interpretation for Java Based Smart Card to Mitigate Logical Attacks”. In: *Proceedings of the International Conference on Recent Trends in Computer Networks and Distributed Systems Security (SNDS)*. Trivandrum, India, Oct. 2012, pp. 185–194. DOI: [10.1007/978-3-642-34135-9\\_19](https://doi.org/10.1007/978-3-642-34135-9_19).

- [BIL11] **Guillaume Bouffard**, Julien Iguchi-Cartigny, Jean-Louis Lanet, “Combined Software and Hardware Attacks on the Java Card Control Flow”. In: *Proceedings of the 10th International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Emmanuel Prouff. Vol. 7079. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Sept. 2011, pp. 283–296. doi: [10.1007/978-3-642-27257-8\\_18](https://doi.org/10.1007/978-3-642-27257-8_18).
- [Bou+11] **Guillaume Bouffard**, Jean-Louis Lanet, Jean-Baptiste Machemie, Jean-Yves Poichotte, Jean-Philippe Wary, “Evaluation of the Ability to Transform SIM Applications into Hostile Applications”. In: *Proceedings of the 10th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Ed. by Emmanuel Prouff. Vol. 7079. Lecture Notes in Computer Science. Leuven, Belgium: Springer, Sept. 2011, pp. 1–17. doi: [10.1007/978-3-642-27257-8\\_1](https://doi.org/10.1007/978-3-642-27257-8_1).


### National Conference Proceedings

- [TB25] Philippe Trébuchet, **Guillaume Bouffard**, “300 secondes chrono: prise de contrôle d’un infodivertissement automobile à distance”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. June 2025.
- [BG18] **Guillaume Bouffard**, Léo Gaspard, “Hardening a Java Card Virtual Machine Implementation with the MPU”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2018.
- [LB16] Julien Lancia, **Guillaume Bouffard**, “Fuzzing and Overflows in Java Card Smart Cards”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2016.
- [BL14a] **Guillaume Bouffard**, Jean-Louis Lanet, “Escalade de privilège dans une carte à puce Java Card”. In: *Symposium sur la sécurité des technologies de l’information et des communications (SSTIC)*. Rennes, France, June 2014.
- [Bou+13b] **Guillaume Bouffard**, Mathieu Lassale, Sergio Ona Domene, Hanan Tadmori, Jean-Louis Lanet, “Intégration d’une politique de flot de contrôle dans un automate de sécurité”. In: *8ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d’Information (SAR-SSI)*. Mont de Marsan, France, Sept. 2013.
- [Ham+12] Samiya Hamadouche, **Guillaume Bouffard**, Jean-Louis Lanet, Bruno Dorsemaine, Bastien Nouhant, Alexandre Magloire, Arnaud Reynaud, “Subverting Byte Code Linker service to characterize Java Card API”. In: *Proceedings of the 7th Conference on Network and Information Systems Security (SAR-SSI)*. Cabourg, France, May 2012, pp. 75–81. ISBN: 978-2-9542630-0-7.
- [Nou+09] Agnès Cristèle Noubissi, Ahmadou Al Khary Sere, Julien Iguchi-Cartigny, Jean-Louis Lanet, **Guillaume Bouffard**, Julien Boutet, “Carte à puce : Attaques et Contremesures”. In: *Majecstic*. 1112. Avignon, France, Nov. 2009.




## A.7 Ph.D. Thesis Reports and Defense Minutes

### A.7.1 Ph.D. Thesis Report by Prof. David Naccache



**Université  
de Limoges**  
Collège Doctoral de Site



**UNIVERSITÉ DE LIMOGES**  
**COLLÈGE DOCTORAL DE SITE**

**FICHE D'ÉVALUATION GÉNÉRALE**  
(à retourner accompagnée du rapport écrit complet signé)

**Pôle Recherche - E.D. 521, Informatique**

**RAPPORT établi par** Monsieur NACCACHE David

**Nom du candidat** : M. Guillaume BOUFFARD

**Titre de la thèse** : Generic approach for protecting Java Card Smart Card against software attacks.  
*Une approche générique pour protéger les cartes à puce Java Card contre les attaques logicielles.*

**I- digne d'être soutenue, en vue du doctorat :**  
**15/10**

**SUR LE FOND**

OUI   
(sans modification)
NON 
OUI   
(avec modification notable)

**SUR LA PRESENTATION**

OUI   
(sans modification)
NON 
OUI   
(avec modification notable)

**II- cette thèse est-elle :**

**- d'un niveau scientifique**

excellent	très bon	bon	satisfaisant
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>


**- d'une présentation matérielle**

excellente	très bonne	bonne	satisfaisante
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**Date :** 8 AOUT 2014

**Le Rapporteur (signature)**  
DAVID NACCACHE  
*David Naccache*

ÉCOLE NORMALE SUPÉRIEURE  
DÉPARTEMENT D'INFORMATIQUE  
45, rue d'Ulm - 75280 PARIS CEDEX 05



Université Panthéon - Assas Paris 11  
Pr. David Naccache



Prof. David Naccache  
 david.naccache@ens.fr +33 7 86 26 76 30

Paris, le 12 août 2014

**RAPPORT SUR LA THÈSE**  
 « A generic approach for protecting Java card smart card against software attacks »  
 Présentée par M. Guillaume Bouffard

La thèse présentée par M Guillaume Bouffard est divisée en 10 chapitres auxquels s'ajoutent une bibliographie, et des annexes.

Dans un premier temps, le candidat expose, de manière très pertinente et fort complète, l'état de la technologie actuelle de cartes à puce (historique, conception, structure et écosystème) et des Java cartes en particulier (la technologie Java et sa variante pour cartes). Suit une description de l'état de l'art en matière d'attaques logicielles sur des cartes (sans et avec post-émission) ainsi qu'un exposé des mesures applicatives, systémiques et dynamiques permettant de prévenir ces attaques.

Les contributions originales de la thèse, clairement identifiées comme telles par l'auteur, débutent au chapitre 6 où l'idée nouvelle d'appliquer les arbres d'attaque à la Java Card est introduite, expliquée, et mise en œuvre. L'objectif de la thèse est ensuite spécifié : d'apporter des solutions à plusieurs problématiques essentielles résultant de la découverte de nouvelles attaques à l'aide de cet outil. Ces objectifs sont formalisés et analysés de manière très pertinente et claire par la candidate. Il est clair, à la lecture de l'exposé, que le candidat, qui est de toute évidence un excellent programmeur, maîtrise parfaitement à la fois le détail et le fonctionnement de machines virtuelles ainsi que leur structure interne.

Le chapitre 7 intitulé « Java card control flow security » fait une jonction très pertinente entre les besoins exprimés et la mise en œuvre des attaques. Ce chapitre analyse l'effet de la modification des opérations de branchement et l'activation de code malveillant au sein de la machine virtuelle par deux moyens différents (via le vérifieur et par un virus). Des automates permettant de protéger le flux de contrôle sont formalisés, présentés et analysés ainsi qu'un moniteur de référence.

Le rapporteur souligne que la mise en œuvre effective de telles protections dans

45, rue d'Ulm, F-75230 Paris Cedex 05 . Tél. : + 33 (0) 1 44 32 20 34 . Fax : + 33 (0) 1 44 32 20 75. e-mail: david.naccache@ens.fr  
 ÉCOLE NORMALE SUPÉRIEURE  
 DÉPARTEMENT D'INFORMATIQUE  
 45, rue d'Ulm - 75280 PARIS CEDEX 05



des produits commerciaux pourrait faire évoluer et basculer positivement l'ensemble de l'industrie de la carte.

Les expérimentations pratiques permettent d'établir le bien-fondé des concepts proposés sont décrites (notons aussi le chapitre 9 qui est excellemment bien écrit). Le rapporteur tient à signaler la très grande originalité et la précision scientifique avec laquelle la candidate analyse et évalue les technologies concernées.

Après avoir défini de manière convenable l'architecture fonctionnelle, l'ensemble des exigences de sécurité et les exigences systémiques dérivées, la thèse se penche sur la sécurisation du processus de l'édition de liens (« linking »). La manière dont ce processus est mis en œuvre est expliquée, des attaques nouvelles et très astucieuses sont expliquées et mises en œuvre. Des idées et des concepts protectifs nouveaux sont mis en place, de manière intégrée et corrélée, afin de contrer ces attaques.

Le chapitre 9, que nous avons déjà mentionné précédemment, est un compte-rendu expérimental très utile qui aide à mieux comprendre l'impact des idées théoriques développées auparavant.

Comme il se doit, un chapitre de conclusion récapitule ce qui a été étudié, et les perspectives du futur en la matière.

Une bibliographie comportant des ouvrages et articles, standards, guides de référence, normes, projets et groupes d'étude, et sites internet, complète le dispositif des références à cette thèse. Un ensemble d'annexes complète et spécifie les documents et les principes latéraux qui ont servi à écrire et discuter la présente thèse.

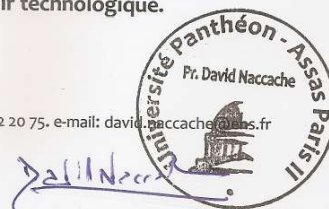
Le style d'ensemble du mémoire est concis, factuel et très agréable : Beaucoup de graphiques et tableaux aident à une bonne compréhension de l'ensemble.

Sur le plan scientifique, cette thèse apporte une vision cohérente, structurée, précise, et détaillée des problèmes de sécurité, de sûreté des cartes Java et de leur compatibilité avec les normes de sécurité existantes, fort intéressante pour le devenir de l'industrie de la carte européenne.

Les choix technologiques à opérer dans le futur sont clairs et innovants. Les analyses qui sont exposées sont précises et détaillées. Qui plus est, la thèse a le mérite d'aller aussi loin que possible sur l'analyse fonctionnelle et organique qui dérive des idées et des innovations présentées.

En conclusion : le mémoire de doctorat présenté par le candidat est un travail impressionnant par son étendue thématique et exemplaire, pour sa clarté et son originalité.

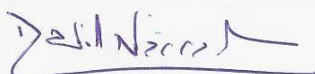
**Ces travaux, répondent absolument à tous les prérequis d'une excellente thèse de doctorat: créativité, pluridisciplinarité, sens du détail et savoir technologique.**



Il s'agit donc d'un mémoire brillant, de qualité scientifique exceptionnelle que le rapporteur classe parmi les **7% meilleures thèses** du domaine, présentées durant les 10 dernières années.

Pour toutes ces raisons, le rapporteur recommande, *sans réserve aucune*, que le candidat soit autorisé à défendre cette excellente thèse.

David Naccache  
Membre de l'IUF et du laboratoire d'informatique de l'Ecole normale supérieure  
Professeur à l'Université Paris II, Panthéon-Assas



ÉCOLE NORMALE SUPÉRIEURE  
DÉPARTEMENT D'INFORMATIQUE  
45, rue d'Ulm - 75230 PARIS CEDEX 05



## A.7.2 Ph.D. Thesis Report by Prof. Peter Ryan

The logo for SNT (Security and Trust) consists of the letters 'SNT' in a bold, black, sans-serif font. A horizontal bar is positioned below the letters, with a red segment on the left and a blue segment on the right.

securityandtrust.lu

Interdisciplinary Centre for Security,  
Reliability and Trust

Luxembourg, September 12<sup>th</sup> 2014

Subject: PhD "A Generic Approach for Protecting Java Card Smart Card  
Against Software Attacks"

Guillaume Bouffard

Examiner's report 12/09/14

The thesis is really quite impressive. The candidate has clearly achieved a complete mastery of the complexities of smart card architectures and associated security mechanisms as well as known classes of attacks. The historical overview is very informative and presented with style and humour. Similarly, the following chapters that present the state of the art in terms of smart card architectures and ways to attack them appears to be highly comprehensive.

I did have a little trouble identifying what exactly are the novel contributions and how significant they are. Sometimes for example, known attacks seem to merge into new ones, or improvements etc. as far as I can tell he seems to have made a number of useful contributions, certainly judging by the number of publications. A summary in the intro of the novel contributions would have been useful I think.

There are some minor, essentially stylistic, problems of English that I will document separately. A rather charming French accent shows through from time to time, for example, the use of "...X is exposed in..." rather than "...X is presented in..."

It seems that an important aspect is his application of FTA to the security aspects of smart cards. Unfortunately though I did not really get a feel for how systematic this was, was it supported by tools for example? How is it related to notions of "attack trees"? It all seems a bit ad hoc. It clearly did throw up some new attacks, but

Interdisciplinary Centre for Security,  
Reliability and Trust  
University of Luxembourg  
4, rue Alphonse Weicker  
L-2721 Luxembourg

T. +352 / 46 66 44 5667  
F. +352 / 46 66 44 5669  
snl@uni.lu  
www.securityandtrust.lu

Établissement public  
Loi du 12 août 2003  
Mémorial A149 du 6 octobre 2003  
TVA Intracom LU 19805732  
N° R.C.S.L. – Luxembourg J20

The logo of the University of Luxembourg features a stylized representation of the letters 'U' and 'L' using vertical bars of varying heights and colors (red, blue, and white).

UNIVERSITÉ DU  
LUXEMBOURG

it wasn't clear what the coverage would be, i.e. how many more classes of attacks would be missed. I realize that this is always a problem with any form of security analysis, but nonetheless some more discussion might have been useful.

It is clear that the candidate has made significant contributions in terms of identifying new attacks and counter-measures in the cat-and-mouse battle over smart card security. These contributions are in my view eminently worthy of the award of a PhD and indicate that M. Bouffard has shown himself capable of performing cutting edge research.

Yours ever,



P Y A Ryan

Professor of Applied Security  
Tel: (+352) 46 66 44 -5667  
Peter.ryan@uni.lu

## A.7.3 Official Report of the Thesis Defense

DOCTORAT  
UNIVERSITE DE LIMOGES



Discipline : Informatique

Sujet :  
"Generic approach for protecting Java Card  
Smart Card against software attacks.

*Une approche générique pour protéger les  
cartes à puce Java Card contre les attaques  
logicielles»*

Membres du jury

M. D. NACCACHE, Professeur des Universités  
M. P. RYAN, Professeur des Universités  
M. POLL Eric, Professeur des Universités  
M. E. PROUFF, Ingénieur-Chef  
M. E. VETILLARD, Ingénieur  
M. J.L. LANET, Professeur des Universités

UNIVERSITE DE LIMOGES

FACULTE DES SCIENCES ET TECHNIQUES

E.D 521  
Sciences et Ingénierie pour l'Information (SII)

**RAPPORT DE SOUTENANCE**

Thèse de

**M. Guillaume BOUFFARD**

Limoges, le 10 octobre 2014

*M<sup>r</sup> Guillaume Bouffard nous a présentés ses travaux sur la protection des cartes Java Card au cours d'un exposé d'une grande qualité pédagogique. Il a su présenter des aspects très techniques de ses travaux d'une manière très claire et convaincante. Il a su montrer au travers de son exposé une excellente maîtrise d'un spectre large de connaissances ce qui a été conforté lors de la séance aux questions.*

*Le jury a apprécié que son travail soit largement diffusé au travers de nombreuses publications.*

*Pour toutes ces raisons, le jury, unanime décerne à M<sup>r</sup> Guillaume Bouffard le titre de docteur de l'université de Limoges avec la mention très honorable.*

Limoges, le  
Nom et Signature du Président

Erik Poll



10/05/1987

Etudiant : M. GUILLAUME BOUFFARD  
 Diplôme : DOCTORAT INFORMATIQUE  
 Titre des travaux : Generic approach for protecting Java Card smart card against software attacks.  
 Une approche générique pour protéger les cartes à puce Java Card contre les attaques logicielles.  
 Ecole doctorale : Sciences et Ingénierie pour l'Information  
 Section CNU : 27 - Informatique  
 Unité de recherche : XLIM - Mathématiques Informatique  
 Directeur : M. JEAN-LOUIS LANET  
 Lieu de soutenance : Faculté des Sciences et Techniques de Limoges -Salle de Conférence XLIM-

La soutenance est publique.

Résultat : *admis*  
 Mention : *Très honorable.*  
 Avis de reproduction : *Favorable*

Membres du Jury

Nom	Qualité	Etablissement	Rôle	Signature
M. PETER RYAN	PROFESSEUR DES UNIVERSITES	Fac.Sc. Tech./Com. Luxembourg	Rapporteur	
M. DAVID NACCACHE	PROFESSEUR DES UNIVERSITES	Ecole Normale Supérieure d'Ulm (Paris)	Rapporteur	
M. JEAN-LOUIS LANET	PROFESSEUR DES UNIVERSITES	UNIVERSITE LIMOGES	Membre	
M. ERIC POLL	PROFESSEUR DES UNIVERSITES	Facul Sciences Pays BAS	Membre	
M. EMMANUEL PROUFF	INGENIEUR	Agence Nat Sécu Systé d'Inf.	Membre	
M. ERIC VETILLARD	INGENIEUR	Cto Div. Smart Card Oracle	Membre	

CADRE A RENSEIGNER PAR LE JURY DE SOUTENANCE

Confidentialité de la thèse :

NON  
 OUI

Corrections demandées:

NON  
 OUI

Jusqu'en (préciser-obligatoirement) :

# Glossary

A | B | C | E | I | O | P | U | W

## A

**ANSSI** National Cybersecurity Agency of France (ANSSI) is a French agency established by decree in July 2009. This national authority is attached to the General Secretariat for Defense and National Security (SGDSN), the body responsible for assisting the Prime Minister in fulfilling his responsibilities in defense and national security. ANSSI is tasked with building and organizing the protection of the Nation against cyberattacks. It thus contributes to strengthening the overall level of cybersecurity and the stability of cyberspace. [1](#), [2](#), [6](#), [13](#), [29](#), [51](#), [52](#), [57](#), [58](#), [60](#), [61](#), [85–90](#), [92](#), [93](#)

## B

**BMC** The Baseboard Management Controller (BMC) is a component that manages controller used in computer devices such as server. It is composed of a microcontroller embedded on the motherboard. [3](#)

## C

**CAP file** The Java Card CAP (*Converted Applet*) file format is used to load Java applications onto the [JVM](#). Each CAP file includes all classes and interfaces defined in a single Java package and can represent either a library or an applet. This file format results from translating a Java class file using the Java Card converter. [15–20](#), [109](#)

**class file** A Java class file is the result of compiling Java source code. It contains Java bytecode that can be executed on the [JVM](#). Each class file defines a single Java class and includes the bytecode for its methods. When an application consists of multiple Java classes, they are typically packaged together in a JAR file, which archives all the class files. [15](#), [17](#), [19](#)

**COTS component** Commercial off-the-shelf (COTS) components refer to ready-made hardware products that are commercially available to the public [[Wik24a](#)]. [7](#), [8](#), [22](#), [31](#), [33](#), [34](#), [36](#), [43](#), [53](#), [54](#), [92](#)

## E

**export file** The Java Card export file contains public [API](#) linking information for all classes in a package. This file is generated during the conversion of Java class files to Java Card [CAP file](#). In the class file, symbols are resolved by their Unicode names. Because storing this information directly would be too memory-intensive for a [SE](#), the Java Card converter translates these Unicode symbols into token-based references using the export file as a reference for all available

public tokens. When a new library that provides public functionalities is converted, the Java Card converter creates an export file that assigns these token values. 15, 18–20

**I**

**In-Order** A CPU pipeline architecture in which instructions are fetched, decoded, and executed sequentially in the order they appear in the program, without reordering for optimization. 35, 36

**O**

**Out-of-Order** A CPU pipeline architecture that allows instructions to be executed in a different order than they appear in the program, enabling performance optimizations by exploiting instruction-level parallelism and reducing idle pipeline stages. 32

**P**

**Protection Profile** A *Protection Profile* is a document used in the Common Criteria certification process to serve as a generic form of a Security Target, created to specify security requirements independent of implementation. It includes threats, security objectives, assumptions, functional requirements, assurance requirements, and rationales [Wik24d]. 4, 6–9, 12, 18, 29, 40, 53, 55–57, 59, 88

**U**

**USART** Universal Synchronous/Asynchronous Receiver-Transmitter (USART), a hardware communication module that supports both synchronous and asynchronous serial communication between devices. 22, 23

**W**

**White-Box Cryptographic implementation** A white-box implementation refers to a cryptographic software design where the attacker is assumed to have full visibility and control over the execution environment, including access to the binary code, memory, and runtime operations. This model presents a significant challenge, as it requires protecting cryptographic secrets against an adversary capable of reverse engineering and dynamic analysis [Cho+02]. 47, 48



## CONTRIBUTIONS À LA SÉCURITÉ DES LOGICIELS EMBARQUÉS DANS LA CHAÎNE DE CONFIANCE

### Résumé

Ce mémoire d'habilitation à diriger les recherches (HDR) porte sur la sécurisation des logiciels dans les systèmes embarqués à travers la notion de chaîne de confiance (*Chain of Trust*, CoT). En 2011, mes travaux ont débuté dans un contexte où la sécurité des opérations sensibles reposait principalement sur la carte à puce, principal exemple de racine de confiance (*Root of Trust*, RoT) matérielle. Ces dispositifs, à l'architecture minimaliste, offraient un haut niveau de sécurité pour une consommation énergétique très faible, au prix de performances limitées. Ces principes ont été étendus aux éléments sécurisés (SE), devenus une référence en matière de RoT matérielle.

À partir de 2016, l'évolution des systèmes embarqués et les besoins croissants en performance ont conduit à la migration d'opérations critiques vers les environnements d'exécution de confiance (*Trusted Execution Environments*, TEE), exécutés sur des processeurs applicatifs. Ces architectures reposent désormais sur une CoT, dont la RoT matérielle constitue le socle.

Ce manuscrit présente mes contributions à l'analyse et au renforcement de la sécurité logicielle dans les différentes couches de cette CoT : RoT matérielle, TEE et environnement riche d'exécution (*Rich Execution Environment*, REE). J'ai d'abord étudié le logiciel embarqué dans les SE (notamment Java Card), ainsi que la sécurité des interfaces et de l'architecture matérielle.

J'ai ensuite étendu cette approche aux TEE, dont les implémentations partagent généralement le processeur avec le REE pour concilier performance et consommation énergétique. Je montre que, comme pour les SE, la robustesse matérielle est essentielle, et j'évalue l'impact des attaques par injection de fautes sur plusieurs TEE.

Enfin, dans les systèmes modernes, l'accès aux TEE ou SE reste souvent restreint pour les développeurs tiers. Les applications sensibles doivent alors s'exécuter dans le REE, considéré comme hostile. J'ai étudié leur vulnérabilité face aux attaques en boîte blanche et exploré des contre-mesures logicielles, telles que l'obscurcissement, pour renforcer la résilience globale de la CoT.

**Mots clés :** chaîne de confiance, sécurité, attaques matérielles et logicielles

---

### Abstract

This habilitation thesis (HDR) focuses on securing software in embedded systems through the concept of a Chain of Trust (CoT). When I began my research in 2011, the protection of sensitive operations primarily relied on smart cards, the main example of a hardware Root of Trust (RoT). These minimalist devices provided high security with very low power consumption, at the cost of limited performance. Their principles were later extended to Secure Elements (SEs), which are now standard hardware RoTs.

From 2016, the growing complexity of embedded systems and increasing performance demands led to a shift of critical operations to Trusted Execution Environments (TEEs) running on application processors. These architectures now rely on a CoT rooted in the hardware RoT.

This manuscript presents my contributions to the analysis and hardening of embedded software across the CoT: the hardware RoT, the TEE, and the Rich Execution Environment (REE). I first investigated SEs, especially Java Card platforms, as well as the security of their interfaces and underlying hardware.

I then extended this work to TEEs, typically implemented by sharing the application processor with the REE to balance performance and energy efficiency. As with SEs, hardware robustness is key. I analyze the impact of fault injection attacks on existing TEEs.

Finally, in modern systems, access to TEE or SE functionality is often limited for third-party developers. Sensitive applications must therefore run in the REE, an untrusted environment. I studied their vulnerability to white-box attacks and explored software-level countermeasures, such as code obfuscation, to improve resilience and enhance the overall security of the CoT.

**Keywords:** chain of trust, security, hardware and software attacks

---