Etude d'attaques matérielles et combinées sur les "System-on-chip"

Fabien Majéric

N°d'ordre NNT : 2018LYSES050

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
Université Jean Monnet – Saint-Étienne

Ecole Doctorale N° 488
Sciences Ingénierie Santé

Spécialité de doctorat :
Microélectronique / Systèmes Embarqués Sécurisés

Soutenue publiquement le 30/11/2018, par :
Fabien Majéric

Etude d'attaques matérielles et combinées sur les « System-on-Chip »

Devant le jury composé de :

Cogniat, Guy,	Professeur des Universités, Université Bretagne Sud	Président
Dutertre, Jean-Max	Maître-Assistant HDR, Ecole Nationale Supérieure des Mines de Saint-Etienne	Rapporteur
Maurine, Philippe	Maître de conférence HDR, Montpellier	Rapporteur
Fischer, Viktor	Professeur des Universités, UJM	Membre du jury
Flottes, Marie-Lise	Chargée de recherche, CNRS	Membre du jury
Bourbao, Eric	Ingénieur, GEMALTO	Encadrant
Bossuet, Lilian	Professeur des Universités, UJM	Directeur de thèse

Table des matières

Table des figures

Liste des tableaux

Glossaire détaillé

ADB *Android Debug Bridge*. En français “passerelle de débogage pour Android”, est un outil en ligne de commande permettant de déboguer Android.

AES *Advanced Encryption Standard*. En français “standard de chiffrement avancé”, aussi désigné par son nom d'origine “Rijndael”, est un algorithme de chiffrement symétrique ayant remporté le concours AES en 2000.

AHB *Advanced High-performance Bus*. En français “bus haute-performance avancé”, est un bus faisant parti de la spécification AMBA d'ARM® permettant des transferts haut débit entre deux modules travaillant à la même fréquence. Il est apparu dans la spécification AMBA 2.

AMBA *Advanced Microcontroller Bus Architecture*. En français “architecture avancée de bus de microcontrôleur”, est une famille de bus informatiques utilisée dans les systèmes sur la puce. AMBA est une marque enregistrée par ARM®.

ANSSI Agence Nationale de la Sécurité des Systèmes d'Information. Service français d'autorité nationale chargé en matière de sécurité des systèmes d'information. À ce titre elle est chargée de proposer les règles à appliquer pour la protection des systèmes d'information de l'État et de vérifier l'application des mesures adoptées.

APB *Advanced Peripheral Bus*. En français “bus de périphériques avancé”, est un bus faisant parti de la spécification AMBA d'ARM® ayant des caractéristiques permettant de communiquer avec les périphériques.

API *Application Programming Interface*. En français “interface de programmation applicative”, est un ensemble normalisé de classes, de méthodes ou de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

ASCII *American Standard Code for Information Interchange*. En français “code américain normalisé pour l'échange d'information”, est une norme de codage de caractères.

ASIC *Application-Specific Integrated Circuit*. En français “circuit intégré propre à une application”, est un micro-circuit électronique intégré spécialisé. En général, il regroupe un grand nombre de fonctionnalités uniques ou sur mesure.

ATB *Advanced Trace Bus*. En français “bus de trace avancé”, est un bus faisant parti de la spécification CoreSight d'ARM®, dédié au débogage implanté en parallèle d'autres

bus. Il permet d'observer les différentes parties du système pendant leur fonctionnement.

AXI *Advanced Extensible Interface.* En français "interface extensible avancé", est un bus faisant parti de la spécification AMBA d'ARM®. Apparu dans la spécification AMBA 3, il permet des transferts simultanés de données entre plusieurs modules travaillant à des fréquences différentes.

BBI *Body-Bias Injection.* En français "injection de tension dans le substrat", est une technique d'injection de fautes utilisée pour attaquer un système électronique sécurisé.

BGA *Ball Grid Array.* En français "matrice de billes", est un type de boîtier de circuit intégré, destiné à être soudé sur un circuit imprimé.

BSC *Boundary-Scan Cell.* En français "cellule Bounary-Scan", est un élément mémoire ajouté à chaque signal primaire d'entrée/sortie d'un composant compatible avec la norme IEEE 1149.1.

BSDL *Boundary-Scan Description Language.* En français "langage de description des tests de continuité", est le langage de description utilisé pour tester des puces électroniques grâce au port TAP de la norme IEEE 49.1.

BSR *Boundary-Scan Register.* En français "registre Bounary-Scan", est le registre résultant du chaînage des différentes cellules Boundary-Scan d'un composant compatible avec la norme IEEE 1149.1.

BYOD *Bring Your Own Device.* En français "apportez vos appareils personnels", est une pratique qui consiste à utiliser ses équipements personnels (téléphone, ordinateur portable, tablette électronique) dans un contexte professionnel.

C-HWA *Cryptographic Hardware Accelerator.* En français "module d'accélération cryptographique matériel", est un co-processeur conçu pour effectuer rapidement des calculs cryptographiques.

CA *Client Application.* En français "application cliente", est une application qui s'exécute dans le « Rich Execustion Environnement » et qui fait office d'interface avec le TEE pour accéder aux fonctionnalités des TA.

CCM *Clock Controller Module.* En français "contrôleur d'horloges", est un module qui, à partir d'horloges de références fournies par un autre module, génère et contrôle les différentes horloges qui serviront pour l'ensemble des modules du système.

CESTI Centre d'Evaluation de la Sécurité des Technologies de l'Information. Un CESTI est un prestataire de service indépendant, spécialisé et impartial, agréé par l'ANSSI, et qui évalue le niveau de sécurité d'un produit en respectant les règles du schéma de certification français.

CISC *Complex Instruction Set Computer*. En français "processeur à jeu d'instruction complexes", est un type particulier d'architecture matérielle de processeur qui se caractérise par un large ensemble d'instructions complexes. Chaque instruction peut effectuer plusieurs opérations élémentaires comme charger une valeur en mémoire, faire une opération arithmétique et ranger le résultat en mémoire.

CMOS *Complementary metal oxide semi-conductor*. Technologie de fabrication de composants électroniques et, par extension, les composants fabriqués selon cette technologie.

CPA *Correlation Power Analysis*. En français "analyse des corrélations de la consommation", est un type d'attaque cryptographique par canaux auxiliaires.

CPU *Central Processing Unit*. En français "processeur" ou "unité centrale de traitement", est le circuit électronique d'un ordinateur qui permet de manipuler et traiter les instructions d'un programme. Ces instructions peuvent être arithmétiques, logiques ou de la gestion d'entrées/sorties.

CRC *Cyclic Redundancy Check*. En français "contrôle de redondance cyclique", est un code de détection d'erreur couramment utilisé dans les réseaux numériques et les dispositifs de stockages mémoires afin de détecter d'éventuelles modifications accidentelles survenues durant un transfert de données.

DES *Data Encryption Standard*. En français "standard de chiffrement des données", est un algorithme de chiffrement symétrique. Considéré aujourd'hui comme obsolète et non-sécurisant par le fait que les puissances de calculs actuelles permettent de retrouver sa clé en un temps raisonnable par simple attaque systématique *i.e.* en testant toutes les valeurs de clef possibles.

DFA *Differential Fault Analysis*. En français "analyse différentielle des fautes", est un type d'attaque cryptographique par perturbation. Elle consiste à retrouver la clef de chiffrement en comparant des résultats corrects avec des résultats erronés.

DMA *Direct Memory Access*. En français "accès direct à la mémoire", est un procédé informatique où les données circulant directement de, ou vers, un périphérique sans l'intervention du microprocesseur grâce à un contrôleur adapté à la gestion de la mémoire.

DPA *Differential Power Analysis*. En français "analyse différentielle de la consommation", est un type d'attaque cryptographique par canaux auxiliaires.

DRAM *Dynamic Random Access Memory*. En français "mémoire volatile dynamique", est une catégorie peu coûteuses de mémoire RAM, principalement utilisées pour la mémoire centrale d'un ordinateur. Cette mémoire nécessite un rafraîchissement régulier des valeurs pour ne pas les perdre d'où son appellation « dynamique ».

DUT *Device Under Test*. En français "système soumis aux tests", désigne le système qui est en train de subir des tests.

DVFS *Dynamic Voltage & Frequency Scaling.* En français "ajustement dynamique de tension et de fréquence", est une méthode de gestion de l'alimentation et de la fréquence de travail utilisée par les systèmes informatiques.

ECB *Electronic Code Book.* En français "dictionnaire de codes", est un mode de chiffrement. La donnée à chiffrer est divisée en sous-parties qui sont chiffrées indépendamment les unes des autres. Le chiffré final est la concaténation de toutes les sous-parties.

EEPROM *Electrically Erasable Programmable Read-Only Memory.* En français "mémoire électriquement programmable ou effaçable à lecture seule", est un type de mémoire non-volatile dont le contenu est fixé lors de sa programmation. Cette mémoire peut être lu à plusieurs reprises et peut être effacée électriquement.

EM Electromagnétique.

FBB *Forward Body Biasing.* Effet obtenu dans les transistors MOSFET lorsque le potentiel du substrat est supérieur à celui de la source.

FIA *Fault injection Attacks.* En français "attaques par injection de faute", sont une variété d'attaques des systèmes cryptographiques électroniques qui consistent en la comparaison de valeurs correctes et erronées afin de retrouver une clef cryptographique secrète. Les valeurs erronées sont obtenues par perturbation contrôlée d'une grandeur physique du système pendant qu'il exécute une étape intermédiaire de l'algorithme cryptographique.

GPC *General Power Controller.* En français "contrôleur principal de puissance", est le contrôleur responsable de la gestion des alimentations du système tout entier.

GPIO *General Purpose Input/Output.* En français "Entrée/Sortie à usage général", sont des ports d'entrée/sortie utilisés dans les microcontrôleurs pour communiquer avec d'autres circuits.

GPU *Graphics Processing Unit.* En français "unité de gestion graphique", est un circuit intégré spécialisé dans la manipulation rapide de données mémoire pour accélérer la création d'images destinées à un système d'affichage.

HDMI *High Definition Multimedia Interface.* En français "interface multimédia haute définition", est une norme et une interface numérique permettant le transfert de données multimédia (audio et vidéo) non compressées en haute définition.

I/O *Input/Output.* En français "Entrées/Sorties", sont des canaux de communication destinés à l'échange d'informations entre un système informatique et l'extérieur.

IoT *Internet of Things.* En français "internet des objets", représente l'ensemble des objets connectés à internet présents dans la vie quotidienne. Cette notion peut s'appliquer à une multitude de domaines comme par exemple la domotique, l'automobile, la téléphonie, la santé, etc.

IP *Intellectual Property*. En français "propriété intellectuelle", désigne les systèmes soumis à la propriété intellectuelle. Il s'agit de principes de systèmes électroniques, de cellules, d'unités logiques, de concepts d'architectures, etc. qui sont réutilisables et soumis aux droits de la propriété intellectuelle.

IP-block *Intellectual Property block*. En français "partie soumise à la propriété intellectuelle", voir IP.

IP-core *Intellectual Property core*. En français "processeur soumis à la propriété intellectuelle", voir IP.

IR Infrarouge. Rayonnement électromagnétique d'une longueur d'onde comprise approximativement entre 7×10^{-9} m et 1×10^{-3} m.

JEDEC *Joint Electron Device Engineering Council*. Aussi appelée *JEDEC Solid State Technology Association*, est un organisme de normalisation des semi-conducteurs, composé de plus de 300 membres dont certaines des plus grosses compagnies de microélectronique.

JTAG *Joint Test Action Group*. Est le nom du groupe de travail qui à mis au point la norme IEEE 1149.1 « IEEE Standard for Test Access Port and Boundary-Scan Architecture ». Ce terme est abusivement utilisé pour désigner le protocole *Boundary Scan* décrit dans cette norme. Aujourd'hui le JTAG désigne à la fois le *Boundary Scan* mais aussi les possibilités de débogage offert par cet interface.

LED *Light-Emitting Diode*. En français "diode électroluminescente", est un dispositif opto-électronique capable d'émettre de la lumière lorsqu'il est parcouru par un courant électrique.

MEB Microscope Electronique à Balayage. Voir SEM.

MET Microscope Electronique en transmission. Voir TEM.

MMU *Memory Management Unit*. En français "unité de gestion mémoire", est un composant responsable de la gestion entre les adresses physiques et virtuelles. Il permet également de contrôler les accès qu'un processeur fait à la mémoire du système dans lequel il est placé.

MPU *Memory Protection Unit*. En français "unité de protection de la mémoire", est un composant permettant de contrôler les accès qu'un processeur fait à la mémoire du système dans lequel il est placé.

NAND *Not-And*. En français "Non-Et", désigne un opérateur logique de l'algèbre de Boole. Pour $(a, b) \in \mathbb{B}^2$, l'opération a NAND b sera FAUX si et seulement si a et b sont VRAI.

NOR *Not-Or*. En français "Non-Ou", désigne un opérateur logique de l'algèbre de Boole. Pour $(a, b) \in \mathbb{B}^2$, l'opération a NOR b sera VRAIE si et seulement si a et b sont FAUX.

NVM *Non Volatile Memory.* En français “mémoire non-volatile”, est une mémoire qui conserve ses données en l'absence d'alimentation électrique.

NW *Normal World.* En français “monde normal”, aussi connu sous le nom de “Rich OS ” ou “Rich Execution Environment”, désigne le système d'exploitation fournissant une multitude de services facilitant le développement d'applications et l'utilisation de fonctionnalités pour divers systèmes mobiles. Il est l'opposé du « Secure OS » *i.e.* le « Trusted Execution Environment »..

OCRAM *On-Chip RAM.* En français “RAM sur la puce”, est une mémoire RAM interne au système sur puce. De faible capacité, elle assiste souvent la mémoire ROM ou des fonctions liées à la sécurité.

OS *Operating System.* En français “système d'exploitation”, est un ensemble de logiciels qui gère l'utilisation des ressources matérielles et logicielles d'un ordinateur.

OTP *One-Time Programmable.* En français “programmable qu'une fois”, se dit pour des mémoires mortes programmables qu'une seule fois. Voir PROM.

PC *Personal Computer.* En français “ordinateur personnel”.

PCB *Printed Circuit Board.* En français “circuit imprimé”, est un support, en général une plaque, permettant de maintenir et de relier électriquement un ensemble de composants électronique entre eux, dans le but de réaliser un circuit électronique complexe.

PFD *Phase Fractional Dividers.* En français “diviseur de phase”, est un système électronique qui fournit un signal de sortie avec une phase qui est sous-multiple de celle donnée par le signal d'entrée.

PIN *Personal Identification Number.* En français “numéro d'identification personnel”, est un code secret composé uniquement de chiffres destiné à authentifier l'utilisateur d'un appareil, système, carte à puce, etc.

PLL *Phase-Locked Loop.* En français “boucle à phase asservie”, est un système électronique qui génère un signal de sortie dont la phase est asservie à celle du signal d'entrée.

PMU *Power Management Unit.* En français “unité de gestion de puissance”, est le module qui fait l'interface entre l'alimentation électrique extérieure et le système. Il est composé des sources d'alimentation, des transformateurs de puissances et des éléments qui les contrôlent.

PoI *Point of Interest.* En français “point d'intérêt”. En analyse par canaux auxiliaires, il s'agit de la zone temporelle d'un signal qui contient une information intéressante pour un attaquant.

PoP *Package-on-Package.* En français “boîtier sur le boîtier”, est une manière de combiner verticalement des circuits intégrés dans un même boîtier afin de gagner un maximum de place.

PROM *Programmable Read-Only Memory*. En français "mémoire programmable à lecture uniquement", est une mémoire non-volatile du type mémoire morte dont le contenu est fixé définitivement lors de son unique programmation. Cette mémoire peut être lue à plusieurs reprises mais ne peut être modifiée.

R&D Recherche & Développement. Désigne l'ensemble des activités entreprises en vue d'accroître et d'utiliser des connaissances pour de nouvelles applications.

RAM *Random Access Memory*. En français "mémoire vive" ou "mémoire à accès direct", est la mémoire informatique qui stocke les données ou instructions fréquemment utilisées par un programme pour augmenter la vitesse d'exécution globale d'un système. Il s'agit d'un type de mémoire volatile.

RBB *Reverse Body Biasing*. Effet obtenu dans les transistors MOSFET lorsque le potentiel du substrat est inférieur à celui de la source.

REE *Rich Execution Environment*. En français "environnement d'exécution principal", est par opposition au TEE, un système d'exploitation fournissant la majorité des services à un système et à ses utilisateurs.

RISC *Reduced Instruction Set Computer*. En français "processeur à jeu d'instructions réduit", est un type particulier d'architecture matérielle de processeurs qui se caractérise par un ensemble réduit d'instructions simples. Chaque instruction effectue une seule opération élémentaire.

RNG *Random Number Generator*. En français "générateur de nombre aléatoire", est un dispositif capable de générer une séquence de nombres n'ayant pas de relations entre eux.

ROM *Read-Only Memory*. En français "mémoire à lecture uniquement", est une mémoire non-volatile du type mémoire morte dont le contenu est fixé définitivement lors de sa construction. Cette mémoire peut être lue à plusieurs reprises mais ne peut être modifiée.

RSA Rivest Shamir Adleman. Nom d'un algorithme de chiffrement asymétrique communément désigné par son acronyme composé du nom de ses inventeurs.

RSB Rapport Signal à Bruit. Il s'agit d'un indicateur sur la qualité d'une information contenue dans un signal mesuré par rapport aux parasites. C'est un rapport entre la puissance du signal désiré avec celle du bruit parasite.

SCA *Side-Channel Attacks*. En français "attaques par canaux auxiliaires", ou "attaques par canaux cachés", sont une variété d'attaques par cryptanalyse des systèmes cryptographiques électroniques qui se basent sur le fait que les variations des grandeurs physiques des systèmes ont un lien avec les valeurs qu'ils manipulent. La mesure de ces grandeurs permet de retrouver les valeurs secrètes.

SCM *Scanning Capacitance Microscopy.* En français "microscopie à mesure de capacitance", est un type de microscopie qui caractérise la surface d'un échantillon en la balayant de très près avec une sonde électrode-aiguille.

SDK *Software Development Kit.* En français "kit de développement logiciel", est un ensemble d'outils logiciels destinés aux développeurs, facilitant le développement d'un logiciel sur une plateforme donnée.

SEM *Scanning Electron Microscopy.* En français "microscopie électronique à balayage", est une technique de microscopie capable de produire des images en haute résolution de la surface d'un échantillon en utilisant le principe des interactions électrons-matière.

SEMA *Simple Electromagnetic Analysis.* En français "simple analyse du rayonnement électromagnétique", est un type d'attaque cryptographique par canaux auxiliaires.

SoC *System on Chip.* En français "système sur puce", est un système complet embarqué sur une seule puce, pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interfaces, ou tout autre composant nécessaire à la réalisation de fonctions désirées.

SOSD *Sum of Squared Differences.* En français "somme de la différence des moyennes des classes à la moyenne globale", est un outil statistique de détection de fuite d'information. Son équation est donnée Ch.3, Section 3.3.2, Eq. 3.6.

SOST *Sum of Squared pairwise T-differences.* En français "somme de la différence des moyennes des classes deux à deux", est un outil statistique de détection de fuite d'information. Son équation est donnée Ch.3, Section 3.3.2, Eq. 3.7.

SPA *Simple Power Analysis.* En français "simple analyse de la consommation", est un type d'attaque cryptographique par canaux auxiliaires.

SPI *Serial Peripheral Interface.* En français "interface des périphériques série", est une norme pour les bus de communication. Elle décrit l'échange de données en mode série synchrone de manière maître-esclave entre les différents éléments impliqués.

SRAM *Static Random Access Memory.* En français "mémoire volatile statique", est une catégorie de mémoire RAM qui utilise des bascules pour sauvegarder ses données qu'elle maintient en mémoire tant qu'elle est alimentée sans avoir besoin de les recharger. Rapide, fiable mais avec un prix de fabrication élevé, elle est souvent utilisée comme registre CPU ou mémoire cache.

SW *Secure World.* En français "monde sécurisé", aussi connu sous le nom de "Secure OS" ou "Trusted Execustion Environnement" (voir TEE).

SWD *Serial Wire Debug.* Est un protocole de débogage proposé dans la solution CoreSight d'ARM. Il s'agit d'une alternative au JTAG.

TA *Trusted Application.* En français "application fiable", est une application qui s'exécute dans le « Trusted Execustion Environnement » moyennant tout un protocole de vérification et de sécurité à son encontre.

TAP *Test Access Port*. En français “port d’accès aux tests”. Il est décrit dans la norme IEEE 1149.1 comme étant l’interface de communication entre les outils de débogage internes et externes. La communication se fait selon un protocole décrit par cette même norme.

TCK *Test ClocK*. En français “horloge de test”, est le signal d’horloge envoyé par la sonde de débogage au TAP pour cadencer le protocole de débogage matériel. Ce signal est décrit par la norme IEEE 1149.1.

TCP *Transmission Control Protocol*. En français “protocole de contrôle de transmissions”, est un protocole de communication.

TDI *Test Data In*. En français “données en entrée de test”, sont les données séries envoyées depuis une sonde de débogage matérielle vers le circuit à tester. Ce signal est décrit par la norme IEEE 1149.1.

TDO *Test Data Out*. En français “données en sortie de test”, sont les données séries retournées par le circuit testé à une sonde de débogage matérielle. Ce signal est décrit par la norme IEEE 1149.1.

TEE *Trusted Execution Environment*. En français “environnement d’exécution de confiance”, est une zone sécurisée et isolée d’autres environnements d’exécutions. Il fonctionne en parallèle du REE.

TEM *Transmission Electron Microscopy*. En français “microscopie électronique à transmission”, est une technique de microscopie capable de produire des images de haute résolution en « transmettant » un faisceau d’électrons à travers un échantillon très mince.

TLB *Translation Lookaside Buffer*. Il s’agit d’une partie de la mémoire utilisée par l’unité de gestion mémoire (MMU) dans le but d’accélérer la correspondance entre les adresses virtuelles et physiques de la mémoire principale.

TMS *Test Mode Select*. En français “sélection du mode de test”, est le signal qui configure le mode dans lequel le TAP fonctionne. Ce signal agit directement sur la machine à état du JTAG. Il est décrit par la norme IEEE 1149.1.

TRNG *True Random Number Generator*. En français “générateur matériel de nombres aléatoires”, est un système qui génère des nombres aléatoires à partir de la mesure de phénomènes physiques.

TRST *Test ReSeT*. En français “réinitialisation du test”, est un signal optionnel qui vide les différents registres et réinitialise la machine à état du TAP. Il est décrit par la norme IEEE 1149.1.

TVLA *Test Vector Leakage Assessment*. En français “vecteur de test pour l’évaluation des fuites”, désigne un ensemble de données construit de tel manière que son traitement par un module particulier produise un maximum de fuites d’informations.

UART *Universal Asynchronous Receiver Transmitter*. En français "émetteur-récepteur asynchrone universel", aussi appelé *port série*. Système qui permet de communiquer des données octet par octet de manière asynchrone.

UID *User IDentifier*. En français "identifiant d'utilisateur", permet d'identifier un utilisateur sur les systèmes d'exploitation Unix et Linux. Cette identifiant est utilisé pour la sécurité du système afin de gérer les droits d'accès à des ressources ou à des domaines.

USB *Universal Serial Bus*. En français "bus universel en série", est une norme qui définit la forme, le connecteur et le protocole d'un bus servant à connecter ou alimenter des systèmes informatiques. Les câbles peuvent être connectés pendant que les systèmes fonctionnent et la communication se fait en série.

XOR *eXclusive-Or*. En français "Ou-Exclusif", désigne un opérateur logique de l'algèbre de Boole. Pour $(a, b) \in \mathbb{B}^2$, l'opération $a \oplus b$ sera VRAIE si et seulement si a est différent de b.

Unités

Unités SI

Nom	Symbole	Grandeur physique
mètre	m	longueur
seconde	s	temps
kilogramme	kg	masse
ampère	A	intensité électrique
kelvin	K	température

Unités dérivées du SI

Nom	Symbole	Grandeur physique	équivalence SI
hertz	Hz	fréquence	s^{-1}
joule	J	énergie	$kg.m^2.s^{-2}$
Volt	V	tension électrique	$kg.m^2.A^{-1}.s^{-3}$

Unités en dehors du SI

Nom	Symbole	Grandeur physique	remarque
Electron-Volt	eV	énergie cinétique	$\approx 1,602176565(35) \times 10^{-19}$ J
Sample per second	S/s	fréquence d'échantillonnage	
Octet	o	Taille de donnée	$\equiv$ 8 bits

Sources : Bureau International des Poids et Mesures (http://www.bipm.org/)

Notations

Symbole	Signification
$\mathbb{N}$	Ensemble des entiers naturels.
$\mathbb{N}^*$	Ensemble des entiers naturels privé de 0.
$\mathbb{Z}$	Ensemble des entiers relatifs.
$\mathbb{R}$	Ensemble des réels.
$\mathbb{B}$	Ensemble de booléens constitué des valeurs 0 et 1.
$\mathbb{E}()$	Espérance mathématique.
σ_X	Variance de X.
$\mathrm{Cov}()$	Covariance.
$H_W()$	Poids de Hamming.
$H_D()$	Distance de Hamming.
$\overrightarrow{\mathcal{H}}$	Champ magnétique (en A.m^{-1}).
$\oplus$	Opérateur logique booléen OU-Exclusif (XOR).
`0x`$<n>$	Notation hexadécimale du nombre $<n>$.
`0b`$<n>$	Notation binaire du nombre $<n>$.
$\varnothing$	Diamètre.
$[\![i, j]\!]$	Intervalle de nombres entiers avec $(i, j) \in \mathbb{Z}^2$ et $i \leq j$.
$(\vec{e_x}, \vec{e_y}, \vec{e_z})$	Base orthonormée de $\mathbb{R}^3$.

- Les termes anglais sont notés en italique.
- Pour une meilleur lisibilité, les valeurs hexadécimales sont notées en espaçant chaque paquet de 2 octets par un tiret '_'. Par exemple, la valeur sur 32 bits '`0x10000000`' sera notée '`0x1000_0000`'.
- Notation des multiples et sous-multiples d'unités :

n	(nano)	10^{-9}
μ	(micro)	10^{-6}
m	(milli)	10^{-3}
c	(centi)	10^{2}
k	(kilo)	10^{3}
M	(Méga)	10^{6}
G	(Giga)	10^{9}

Définitions

Rappel des définitions données dans le contexte de ce manuscrit.

Définition 1. – **Attaque :** *tout acte perpétré sur un système électronique embarqué visant à dégrader son niveau de sécurité.* (Donnée page 6).

Définition 2. – **Invasif :** *un chemin d'attaque est dit invasif lorsqu'il est nécessaire de modifier de manière irréversible un circuit pour pouvoir l'attaquer. Cette modification est destructive, elle perturbe le fonctionnement normal de ce circuit et peut éventuellement le mettre hors-service* (Donnée page 18)

Définition 3. – **Semi-invasif :** *un chemin d'attaque est dit semi-invasif lorsqu'il consiste à ne modifier qu'une partie du système. La modification est irréversible mais laisse le circuit fonctionnel.* (Donnée page 18)

Définition 4. – **Non invasif :** *un chemin d'attaque est non-invasif lorsqu'il n'affecte pas l'intégrité physique du circuit attaqué. Ce dernier reste totalement fonctionnel.* (Donnée page 18)

Définition 5. – **Globale :** *on dit qu'une attaque est globale lorsque l'action produite par le chemin utilisé concerne l'ensemble du système.* (Donnée page 18)

Définition 6 – **Locale :** *on dit qu'une attaque est locale, lorsque l'action est produite sur la fonctionnalité attaquée et n'affecte pas le reste du système.* (Donnée page 18)

Définition 7 – `<mute>`: *on dit qu'un système est dans l'état* `<mute>` *lorsqu'il ne répond pas aux sollicitations qui lui sont faites sur ses ports I/O.* (Donnée page 74)

Définition 8 – `<fauté>`: *on dit que le résultat d'une opération est* `<fauté>` *lorsque la valeur obtenue est différente de celle retournée durant le fonctionnement normal du système.* (Donnée page 74)

Résumé

Jusqu'à présent, les applications nécessitant un haut niveau de sécurité (paiement, communications, identification, etc.) utilisaient des systèmes exclusivement dédiés tels que les cartes à puce qui intègrent une multitude de contre-mesures logicielles et physiques. Avec la démocratisation des smartphones et autres objets connectés, la manipulation des données sensibles tend à être déléguée à des systèmes plus complexes communément appelés systèmes sur puce, ou en anglais *Systems on Chip* (SoC).

L'intérêt de la communauté de la sécurité numérique dans ce domaine s'est essentiellement focalisé sur les menaces logicielles, améliorant sans cesse le niveau de protection. Cependant, l'exploitation de ce vecteur d'attaque devenant de plus en plus difficile, il est fort probable que les attaques matérielles se multiplient. Par conséquent, il est primordial d'étudier ces dernières afin d'anticiper la menace qu'elles représentent. La sophistication de l'architecture et la rapidité d'évolution des technologies embarquées dans les SoC, justifient la mise en place d'une méthodologie adaptée pour évaluer efficacement leur niveau de sécurité.

C'est dans ce contexte que le travail présenté dans cette thèse propose l'étude de la faisabilité de cette catégorie d'attaques ainsi qu'un aperçu de leur impact sur la sécurité de ce type de systèmes. Alors que les architectures élaborées accroissent la difficulté de mise en place d'attaques physiques, elles augmentent également la surface d'attaque. Une première étude analyse les chemins d'attaques afin de déterminer les grandeurs physiques exploitables les plus pertinentes. Cette étape conduit, dans un deuxième temps, à l'élaboration de règles génériques pour l'évaluation sécuritaire des SoC présents sur le marché. Celles-ci combinent diverses techniques en partie déjà utilisées dans le domaine de la carte à puce. L'ensemble de ce travail s'appuie sur plusieurs cas d'études relatifs à divers modules caractéristiques de la sécurité des systèmes sur puce actuels. Tous les résultats observés aboutissent aux mêmes constatations : la complexité inhérente aux SoC n'est pas suffisante pour les protéger contre les attaques matérielles et l'implémentation des sécurités dans ces systèmes doit se faire sans se reposer sur cette propriété.

Abstract

Until now, applications requiring a high level of security (payment, communications, identification, etc.), exclusively used dedicated systems, such as smart cards, that integrate a multitude of software and hardware countermeasures. With the democratization of smartphones and other connected devices, sensitive data now tend to be processed by more complex systems, commonly called *System on Chip* (SoC).

In this field, the digital security community has mainly focused on software threats ; constantly working to improve the level of protection. Since the exploitation of this attack vector is becoming more and more difficult, it is most likely that the number of hardware attacks will increase. Therefore, it is essential to study these attacks in order to anticipate the threat they represent. The sophisticated architecture and the rapidly changing technologies embedded in the SoC justify the implementation of an adapted methodology, to effectively evaluate their level of security.

In this context, this thesis examines the feasibility of this type of attacks and their impact on the security of these systems. While rich architectures increase the difficulty of setting up hardware attacks, they also increase the attack surface. Our study starts by analyzing the attack paths in order to determine the most relevant exploitable physical quantities. This has led to the development of a generic procedure for the security evaluation of SoCs on the market. This method combines various techniques that are already applied to smart cards. This entire work is based on several case studies related to various embedded modules characteristic of the security in current systems-on-chips. All the observed results lead to the same observations : the inherent complexity of SoCs is not sufficient to protect them against hardware attacks. The implementation of security in these systems must be done without relying on this property.

Introduction

Contexte

Depuis les travaux de Biham et Shamir [29, 30] et ceux de Kocher [78, 79] dans les années 90, le domaine des attaques matérielles est largement étudié par la communauté de la sécurité numérique [21, 34, 11, 10, 13, 87]. L'émergence perpétuelle de nouvelles attaques et l'élaboration de contre-mesures adaptées ont porté au plus au niveau la sécurité des systèmes électroniques dédiés destinés à manipuler des données sensibles, tels que les cartes à puces.

Pour faire face à la problématique de commercialisation de produits fiables et sécurisés, les autorités ont mis en place des schémas de certification. En se basant sur les différentes études de la communauté scientifique, les produits manipulant des informations sensibles et destinés à être commercialisés doivent passer une série de tests afin de garantir la conformité de leur sécurité avec leur cahier des charges [1]. En France, c'est l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI) qui délivre des certificats aux systèmes sécurisés en fonction du niveau de leur résistance et des services qu'ils proposent. Pour évaluer ce niveau, ces derniers sont soumis à une série de tests réalisés par des entités indépendantes : les Centres d'Evaluation de la Sécurité des Technologies de l'Information (CESTI). Les tests comportent une partie consacrée aux attaques matérielles et les évaluateurs ont dû se maintenir à l'état de l'art pour que les évaluations restent pertinentes avec la réalité des menaces. Des échelles de notation, des normes et des séquences de tests adaptées font désormais partie du processus de certification des systèmes matériels embarqués « simples » [2].

Cependant, avec la démocratisation des smartphones et autres objets du même type, la manipulation de données sensibles dans les systèmes embarqués n'est plus exclusivement réservée à des composants sécurisés dédiés mais tend à être en partie ou totalement déléguée à des systèmes polyvalents plus complexes communément appelés *System on Chip* (SoC). Ces derniers embarquent une quantité de modules spécialisés qui exécutent différentes tâches en parallèle. La problématique soulevée ici concerne l'évaluation du niveau de sécurité de ces mécanismes. Aujourd'hui, il n'existe pas de schéma de certification pour de tels systèmes et malheureusement la littérature actuelle ne propose que très peu de solutions d'évaluations parfois non-abouties. Par conséquent, les méthodes d'évaluation classiques doivent être ajustées.

C'est dans ce contexte que les travaux présentés dans ce manuscrit étudient la mise en

1. https://www.ssi.gouv.fr/administration/produits-certifies/
2. https://www.ssi.gouv.fr/administration/produits-certifies/cc/produits-certifies-cc/

place d'attaques matérielles sur des systèmes complexes embarqués pour, d'une part tester la faisabilité de telles d'attaques sur ces mécanismes et, d'autre part, esquisser de nouvelles méthodes d'évaluation. La méthodologie de test résultante sera appliquée sur des systèmes complexes embarqués représentatifs du marché actuel.

Objectif

Le premier objectif de ce travail est de s'inspirer des attaques matérielles déjà existantes dans le domaine la sécurité numérique des circuits, tels que les cartes à puce, et d'étudier leur applicabilité sur des processeurs embarqués plus complexes.

De par leur architecture, ces processeurs offrent de nouvelles sécurités et de nouveaux chemins d'attaques. C'est pourquoi une deuxième partie de ce travail consiste à l'évaluation de ces sécurités et à la recherche de ces chemins afin d'analyser les impacts qu'ils peuvent avoir sur la sécurité globale du système.

Contribution

Les principales contributions des travaux présentés dans ce manuscrit de thèse sont :

- Développement d'une méthodologie de test générique pour appréhender les systèmes complexes embarqués. Cette méthode est appliquée tout au long de ce manuscrit et présente la démarche à aborder face à des systèmes complexes.
- Adaptation de bancs *side-channel* et injection de fautes au contexte des SoC. Cette description présente la structure minimum des bancs automatisés nécessaires pour aborder des évaluations sur des systèmes complexes.
- Guide d'ajustement des paramètres. Ce guide synthétise les observations que nous avons faites durant nos expérimentations au sujet de l'ajustement des paramètres de la grandeur physique retenue pour nos expérimentations. Il donne des méthodes pour définir certains d'entre eux (ex. localisation temporelle des signaux).
- Mise en place d'une méthode de détection de fuites d'information. La combinaison de méthodes statistiques et de traitement du signal a permis de détecter un accélérateur cryptographique avec un faible rapport signal à bruit.
- Nouvelle utilisation détournée du débogage matériel comme moyen d'injection de fautes.
- Mise en évidence de l'applicabilité d'attaques sur des systèmes complexes (accélérateur cryptographique, système d'exploitation isolé, *Secure boot*, mécanismes de verrouillage, etc.)

Organisation du manuscrit

Outre cette introduction et la conclusion finale, ce document est composé de 5 chapitres.

Le Chapitre 1 présente brièvement les systèmes embarqués polyvalents afin d'apprécier la complexité de ces architectures et les défis qu'elles amènent pour les évaluations de

sécurité. Dans ce chapitre, nous exposons également les principales attaques matérielles applicables aujourd'hui sur les systèmes sur puce. Puis, nous décrivons les différents modules proposés par les concepteurs de SoC pour répondre aux divers besoins de sécurité. Ce sont ces mêmes modules qui feront l'objet de nos études, à savoir les processeurs, les modules sécurisés cryptographiques, les nouvelles architectures sécurisées et les systèmes de débogage matériel.

Le Chapitre 2 décrit la méthode et les outils que nous avons développés durant cette étude pour pouvoir expérimenter des attaques matérielles sur des processeurs complexes embarqués. Dans ce chapitre, une réflexion est faite sur les grandeurs physiques que nous devons considérer dans le contexte de notre étude. Une seconde partie propose un guide pour définir les paramètres liés aux grandeurs sélectionnées. Enfin nous expliquons la composition des bancs de mesures et d'injection de fautes que nous avons mis en place pour effectuer nos expérimentations.

Le Chapitre 3 se focalise sur les implémentations cryptographiques logicielles et matérielles que l'on peut trouver dans les SoC. A travers un chiffrement AES, tantôt exécuté par le processeur principal, tantôt par un co-processeur cryptographique, nous appliquons les méthodes développées dans cette étude pour évaluer la faisabilité de mettre en place des attaques matérielles sur ces modules.

Le Chapitre 4 étudie la possibilité d'attaquer les environnements d'exécution de confiance basés sur les nouvelles architectures sécurisées. Les *Secure Boot*, l'extention de sécurité TrustZone® d'ARM® et les *Trusted Execution Environment* sont les solutions sécuritaires les plus répandues dans les systèmes embarqués actuels. Dans les sections qui leurs sont consacrées, chaque solution est détaillée ainsi que les différentes méthodes d'attaque auxquelles elles ont été soumises.

Enfin, dans le Chapitre 5, nous nous intéresserons aux principaux systèmes de débogage matériel existant dans les systèmes complexes embarqués. En analysant les sécurités mises en place sur ces outils de débogage, nous mettrons en lumière les différents moyens pour les contourner. Dans ce chapitre, nous présenterons une méthode permettant de déverrouiller un type de sécurité liée au débogage, puis comment ce dernier peut se révéler être un puissant outil d'attaques matérielles.

Chapitre 1

Sécurité des systèmes complexes embarqués

Ce chapitre place le contexte des travaux présentés dans ce manuscrit. Au travers de celui-ci, nous présentons brièvement les systèmes embarqués multitâches pour mettre en avant leur complexité. Puis, en énumérant les différents chemins d'attaques matérielles envisageables sur les SoC, nous considérons l'exploitabilité de ceux-ci et nous donnons les principales attaques matérielles existantes à ce jour. Enfin, dans la dernière partie, nous décrivons les différentes solutions proposées par les concepteurs de SoC pour répondre aux besoins de sécurité. Ce sont ces solutions qui seront étudiées dans les chapitres suivants.

Sommaire du chapitre

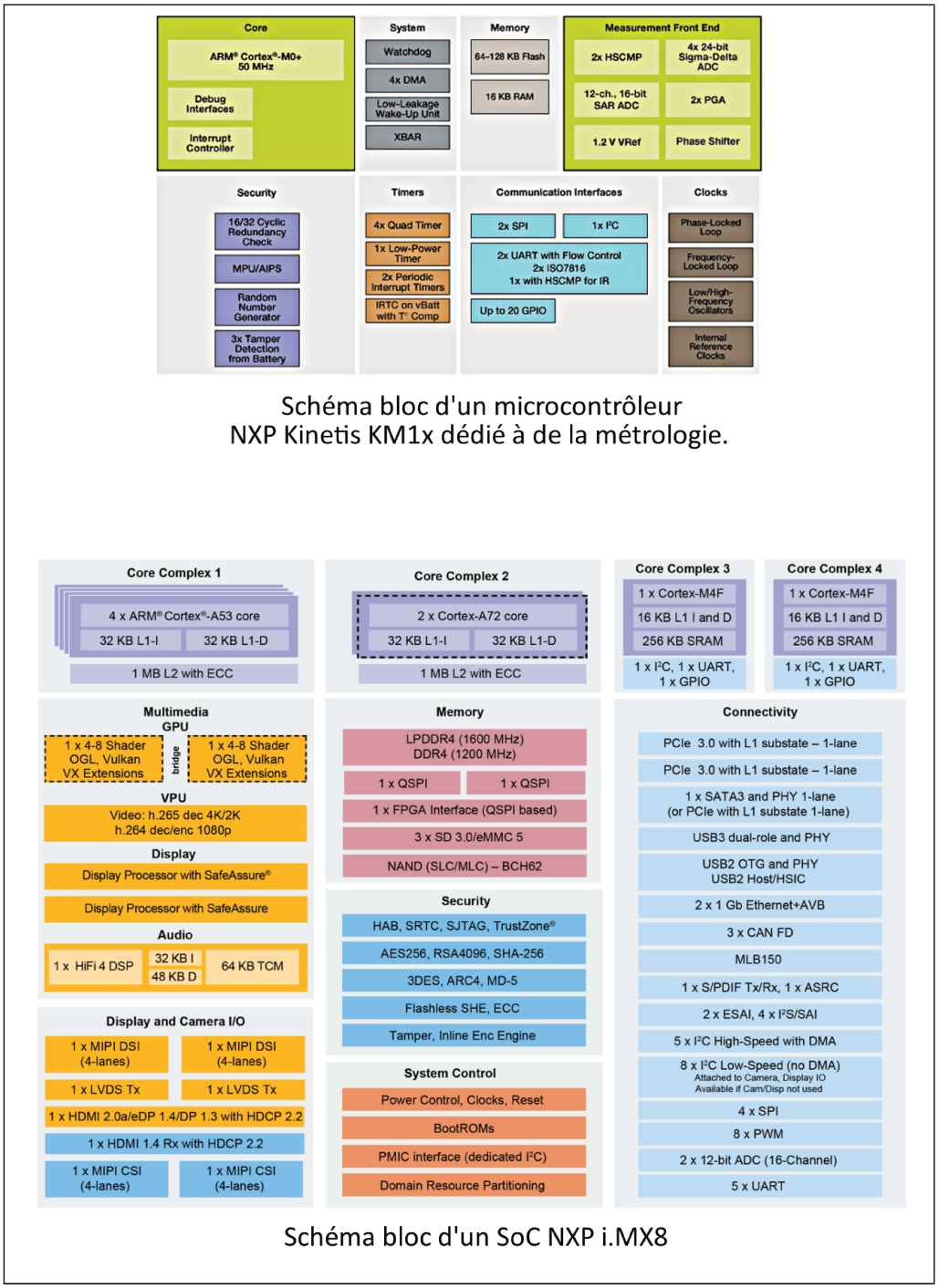


FIGURE 1.1 – Comparaison de la complexité des schémas blocs simplifiés d'un microcontrôleur et d'un SoC [3]

3. Source : http://www.nxp.com

1.1 Les SoC : un système complexe et polyvalent

Un système sur puce, souvent désigné dans la littérature scientifique par le terme anglais *System on a Chip* (SoC), est un système complet embarqué sur une seule puce, pouvant comprendre tous les composants d'un ordinateur. Il peut accomplir des tâches numériques, analogiques ou un mixte des deux grâces aux fonctions qu'il intègre. Ces systèmes sont conçus pour optimiser les rapports entre fonctionnalités, encombrement, performances et dans certains cas consommation électrique. L'expression « système sur puce » traduit plus une idée générale de haut degré d'intégration de différents systèmes sur une même puce qu'une définition technique à proprement parler. Il s'agit du niveau d'intégration supérieur à celui des microcontrôleurs [146]. A titre d'exemple, la figure 1.1 met en avant la différence d'échelle entre les deux systèmes. Elle détaille également les principaux sous-systèmes intégrés dans les SoC que l'on peut trouver. Ces derniers sont décrits dans l'Annexe A de ce document. Dans ce chapitre, nous nous concentrerons uniquement sur les modules impliqués dans la sécurité.

1.1.1 Des architectures hétérogènes et complexes

Alors que les microcontrôleurs dédiés à un type de tâche en particulier n'ont, la plupart du temps, qu'un microprocesseur, quelques kilooctets de RAM, les SoC embarquent plusieurs processeurs puissants et sont prévus pour traiter une multitude de tâches. Les premiers sont des systèmes conçus pour des usages spécifiques, alors que les seconds sont conçus pour êtres rapidement adaptés à de nouvelles fonctionnalités grâce à leur architecture modulaire hétérogène. En effet, au lieu de concevoir entièrement un système adapté à une nouvelle utilisation, les concepteurs réutilisent les architectures en ajoutant des modules dédiés à cette utilisation. Ceci rend ces systèmes suffisamment modifiables et polyvalents pour pouvoir accomplir un large panel de tâches et être intégrés rapidement dans différents appareils embarqués. De cette manière, une production à très grande échelle est envisageable, générant ainsi des économies de temps et de coût de fabrication. Ces avantages sont non négligeables dans un marché très concurrentiel.

1.1.2 Des systèmes modulaires basés sur la propriété intellectuelle

Pour augmenter les rendements, il est très courant pour les concepteurs d'utiliser des parties de systèmes dont les principes ont déja étés éprouvés par d'autres. C'est un modèle commercial très répandu dans l'industrie des SoC selon lequel des compagnies commercialisent des concepts électroniques soumis à la propriété intellectuelle (IP-block, IP-core ou tout simplement IP) pour que les concepteurs n'aient à ajouter que ce dont ils ont besoin. Avec 95% de part de marché dans les processeurs de *smartphones*[5] et plus de 90 milliards de puces intégrant ses IP-block en 2015 (voir Fig. 1.2), l'entreprise ARM® a confirmé sa position de leader mondial, imposant par la même occasion une certaine homogénéité dans les architectures et les spécifications [134]. Ainsi par exemple, pour un système sur puce donné, si une plus grande puissance de calcul est nécessaire, il suffira aux ingénieurs de remplacer l'IP-core du microprocesseur par une IP plus performante en conservant les

4. Source : https://www.arm.com
5. https://www.arm.com/markets/mobile

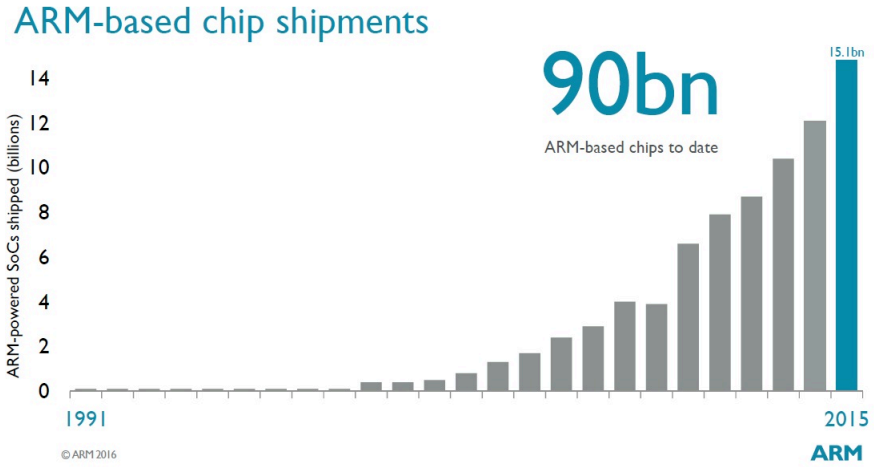


FIGURE 1.2 – Puces intégrant des IP ARM® de 1991 à 2015 [4]

autres déjà présentes (ex. les unités de gestion de l'alimentation, les arbres d'horloges, les différents contrôleurs DMA, USB, HDMI, UART, etc.). Ceci permet une économie considérable en recherche développement. Pour plus de détails au sujet des processeurs ARM®, se reporter à l'Annexe A.2 du présent document.

1.1.3 Des dispositifs gérés par systèmes d'exploitation

Pour gérer l'ensemble des IP-block et le matériel présents dans les SoC, les concepteurs d'appareils font appel à différents systèmes d'exploitation ou *Operating System* (OS). En plus de permettre la gestion du matériel, les OS tels que Android, iOS ou Windows CE ajoutent une certaine portabilité aux applications devant s'exécuter sur les SoC. Cette portabilité est nécessaire au modèle économique des objets complexes connectés tels que les *smartphones*. En effet, d'une part, l'OS offre à la majorité des fabricants d'appareils mobiles une solution « clé en main » qui leur permet de gérer les fonctions de leurs appareils sans avoir à tout développer pour chaque appareil (ex. télécommunications, répertoire, caméra, GPS, lecteur multimédia, etc.) et d'autre part, il apporte aux développeurs une solution complète pour créer des applications qui pourront se déployer sur une large gamme d'appareils. D'un coté cela représente un gain de temps considérable pour les constructeurs d'appareils mobiles et de l'autre cela fournit un support permettant à une large communauté de développeurs de proposer diverses applications sans avoir besoin de se soucier précisément des plateformes sur lesquelles elles s'exécuteront. Pour illustrer ce modèle commercial, l'exemple le plus flagrant est l'OS Android et la plateforme Google Play [6] sur laquelle tout le monde peut déposer ou télécharger des applications. Aujourd'hui, 80% des fabricants de *smartphones* ont intégré Android à leurs appareils.

6. https://play.google.com

1.1.4 Des systèmes attisant les convoitises

La faible consommation énergétique, la polyvalence, la portabilité et les coûts réduits justifient le fait que les SoC soient très répandus dans l'électronique embarquée et particulièrement dans les industries des *smartphones*. Cette situation soulève alors la problématique de sécurité des données. Chaque jour sur la planète, des centaines de millions de ces systèmes manipulent une quantité phénoménale d'informations. L'exploitation de ces données suscite l'intérêt des attaquants et, dans le contexte de cette étude, nous définissons :

Définition 1. – **Attaque :** *tout acte perpétré sur un système électronique embarqué visant à dégrader son niveau de sécurité.*

Cette définition implique que les attaques peuvent cibler les données de tous les acteurs de cette technologie, autrement dit :

- **les utilisateurs** qui sont des personnes physiques ou des entités telles que des entreprises. Ils utilisent cette technologie pour échanger ou stocker des informations plus ou moins sensibles, comme par exemple des identifiants, des mots de passe ou des documents confidentiels. Ce sont ces données qui seront les cibles d'attaques.

- **le commerce numérique.** Certaines applications proposent des services payants. Tout naturellement, des personnes vont essayer de s'affranchir de payer ou pire, de tirer profit en distribuant le produit proposé de manière illégale. Le vol et le recel de donnés compromettent directement le modèle économique proposé par les SoC. L'exemple typique est celui des applications proposant des contenus multimédia. Une transaction doit être uniquement valable pour un média et un utilisateur donné. Si ce dernier parvenait à dupliquer le média qu'il venait d'acheter et, qu'ensuite, il le distribuait, cela aurait un impact sur les revenus attendus par le créateur de la source.

- **les développeurs et concepteurs.** Ces systèmes embarquent des solutions technologiques et des données logicielles qui sont soumises au secret industriel. L'accès à ces secrets par une tierce personne compromet notamment les investissements en recherche et développement (R&D) réalisés par une entreprise. Si un attaquant parvient à voler un secret industriel, il peut alors dupliquer, voir améliorer un produit en s'affranchissant des coûts de R&D investis par la première entreprise. C'est une concurrence totalement déloyale que le système de propriété intellectuelle tente de réguler, mais qu'il vaut mieux prévenir en sécurisant les informations.

Ce travail étudie les attaques appliquées aux systèmes complexes embarqués au sens de la définition 1. En particulier les attaques matérielles et combinées que nous décrivons dans la section 1.2 suivante.

1.2 Attaques matérielles et combinées sur les SoC

Les attaques sur les systèmes électroniques embarqués sont un sujet en perpétuelle évolution. Elles peuvent se décliner sous diverses formes en fonction du type de donnée manipulée et de la technologie utilisée. La section 1.2.1 présente les différents types d'attaques existantes. Les section 1.2.2 et 1.2.3 se focalisent respectivement sur les attaques matérielles et les attaques combinées. Enfin, la section 1.2.4, liste les différents chemins d'attaques exploitables. Dans cette section, on discutera notamment des difficultés d'exploiter ces derniers dans les SoC.

1.2.1 Taxonomie des attaques

Comme représenté par la figure 1.3, les attaques sur les systèmes électroniques embarqués peuvent se répartir selon trois catégories principales. En fonction de leurs objectifs, de la donnée ciblée, différentes techniques sont utilisées. Notre étude se concentre sur les attaques matérielles et les attaques combinées.

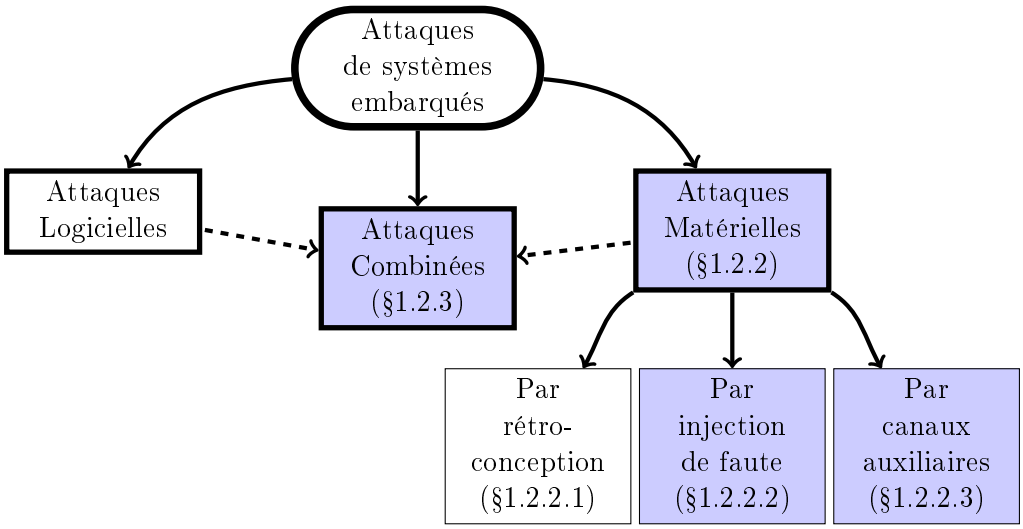


FIGURE 1.3 – Les principaux types d'attaques sur systèmes embarqués. Sont notées en bleu celles sur lesquelles se focalise notre travail. Les sections qui les décrivent sont désignées par (§x.x.x.x).

1.2.2 Les attaques matérielles

Les attaques matérielles ou attaques physiques peuvent être regroupées en trois grandes familles d'attaques :

1. **La rétroconception.**
2. **Les attaques par injection de faute (FIA).**
3. **Les attaques par canaux auxiliaires (SCA).**

1.2.2.1 La rétroconception

La rétro-ingénierie ou rétroconception est un ensemble de techniques qui vise à étudier un système déjà existant. Ces méthodes peuvent être appliquées à n'importe quel type de produit et ont pour objectif de constituer un ensemble d'informations sur ce dernier. Dans le cadre des systèmes électroniques, la rétro-ingénierie peut être utilisée de deux manières :

- De manière défensive. En effet, la rétroconception peut être effectuée par une entreprise, soit pour faire une analyse de risque de défaillance et valider ses propres produits, soit pour s'assurer que les différentes solutions implantées dans une technologie concurrente ne sont pas les siennes.

- De manière offensive. Dans ce cas la rétro-ingénierie est utilisée pour attaquer un produit au sens de la définition 1. Elle est effectuée par une personne ou entité étrangère à la conception d'un produit mais désireuse d'acquérir des informations auxquelles elle ne devrait pas avoir accès. Avec ces informations, une entreprise concurrente peu copier ou améliorer un produit en s'affranchissant de négocier avec l'entreprise propriétaire de celui-ci. Ces informations peuvent également aider à la préparation d'une attaque plus sophistiquée. En effet, en effectuant de la rétroconception, un attaquant peut définir les différents chemins exploitables pour ses objectifs.

Pour effectuer la rétroconception d'un produit et obtenir l'ensemble des informations à son sujet, il est souvent nécessaire de faire appel à des laboratoires spécialisés possédant les compétences et le matériel adéquat. Nous avons dû effectuer quelques étapes de rétroconception afin d'amasser suffisamment d'informations pour pouvoir mettre en place et paramétrer nos attaques. Nous allons expliquer brièvement en quoi cela consiste.

Le type de rétroconception que l'on peut effectuer aujourd'hui sur les systèmes électroniques est largement présenté dans les travaux de [139, 142, 145]. Le principe commun de ceux-ci est de rassembler un maximum de données pour pouvoir les croiser et obtenir un ensemble cohérent concernant une information ciblée. Toutes les sources d'informations disponibles sont exploitées et souvent dans cette ordre : documentation, observation et imagerie, analyse et extraction de caractéristiques.

La documentation. Les informations disponibles à moindre effort sont le point de départ d'une rétroconception. Ainsi, la documentation officielle, les manuels d'utilisation et les schémas de câblage permettent dans un premier temps de comprendre comment le système fonctionne. Dans le contexte des SoC, le fabricant doit permettre aux développeurs d'appareils de comprendre et tester les fonctionnalités des systèmes qu'il propose. Par conséquent, pour une grande partie des SoC du marché, il existe une documentation, des kits de développement et une communauté d'entraide. Les sources d'informations proviennent principalement d'Internet, des sites officiels, des forums et les communautés de développeurs.

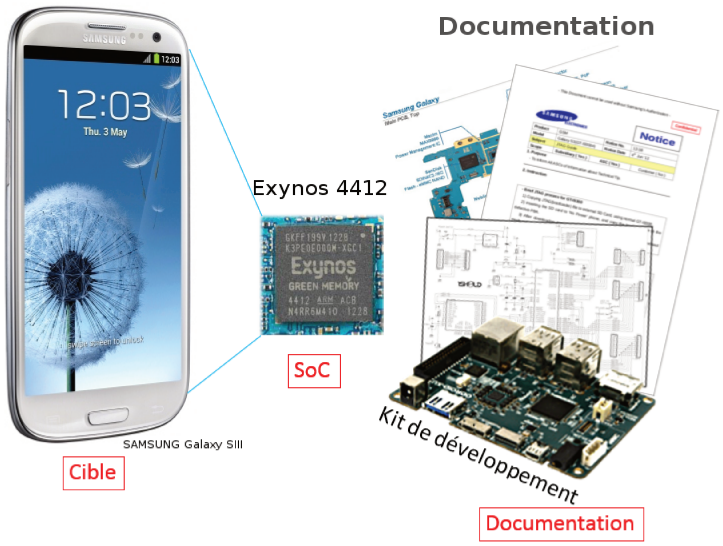


FIGURE 1.4 – Recherche de documentation et d'informations sur le SoC ciblé.

Prenons l'exemple illustré par la figure 1.4 d'un attaquant qui aurait pour cible le SoC Exynos 4412, implanté dans les mobiles Samsung Galaxy SIII [7]. La documentation officielle peut être restreinte. Cependant, en cherchant des informations provenant de sources alternatives, un attaquant pourra se constituer une base d'informations conséquente à propos de cet appareil et de son SoC :

- Commencer par parcourir le site officiel du constructeur dans lequel il pourra trouver de nombreux documents tels que le manuel de référence.

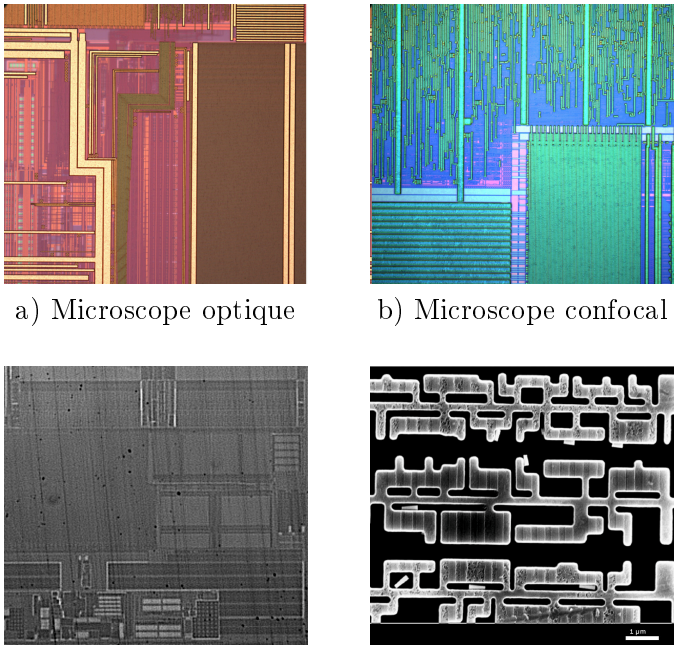
- Parcourir d'autres sites, plus ou moins légaux, qui fournissent des documents techniques comme les schémas de câblage, les références de produits associés (les mémoires externes ou des kits de développement), etc.

- Acquérir des kits de développement tels que la carte ODroid-X [8] permettant de tester les fonctionnalités du SoC. Avec ces kits, l'attaquant pourra prototyper ses attaques.

- Participer à des forums et communautés d'entraide tels que l'XDA-Developers [9] ou le Forensics Wiki [10]. Ceux-ci lui permettront d'échanger avec les développeurs du monde entier, de poser des questions et voir les différents points de vue ainsi que solutions proposées. Les cartes de développement sont souvent l'objet de ce type de forums.

7. https://www.samsung.com/uk/smartphones/galaxy-s3-i9300/GT-I9300ZKDBTU/
8. https://www.hardkernel.com/main/products/prdt_info.php?g_code=G133999328931
9. https://forum.xda-developers.com/galaxy-s3/
10. https://www.forensicswiki.org/wiki/JTAG_Samsung_Galaxy_S3_(SGH-I747M)

L'imagerie. A défaut d'avoir le schéma interne d'une puce, un attaquant essaye d'identifier les différentes parties d'un système grâce à des techniques d'imagerie. Pour cela, il utilise principalement l'imagerie microscopique. Il existe diverses technologies qui présentent des avantages et des inconvénients en fonction de la situation. A titre d'exemple, la figure 1.5 donne le type d'images que l'on peut obtenir avec les principales techniques utilisées dans la rétroconception de circuits électroniques. Le détail de ces techniques n'est pas essentiel pour la compréhension de nos travaux, mais nous invitons néanmoins les lecteurs intéressés à consulter l'annexe C qui les aborde plus en détail. Remarquons que dans l'ensemble, ces techniques ne sont accessibles qu'à des laboratoires spécialisés. Dans la plupart des cas pour lesquels la résolution d'observations est suffisante pour observer les portes logiques d'un SoC, il est nécessaire de préparer l'échantillon de manière invasive. Cette préparation utilise un outillage spécifique et nécessite un savoir-faire particulier [26].



a) Microscope optique b) Microscope confocal

c) Infrarouge d) SEM

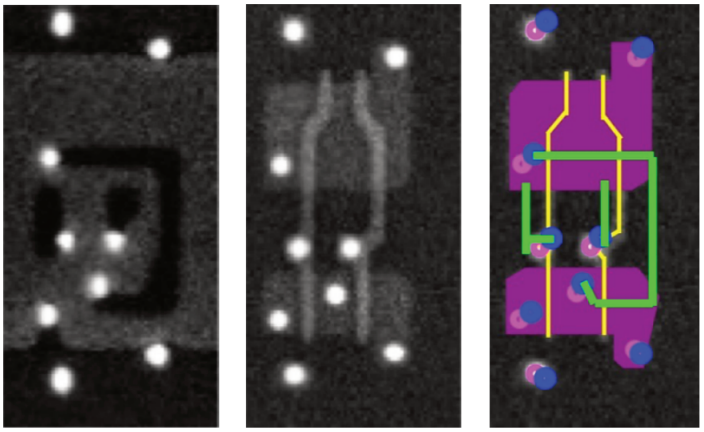
FIGURE 1.5 – Images issues des principales techniques de microscopie utilisées dans la rétroconception de circuits intégrés

L'analyse side-channel. En exploitant la propriété selon laquelle l'activité physique d'un système électronique est liée à son activité logique, il est envisageable de rétroconcevoir une partie de l'implémentation de divers processus s'exécutant sur celui-ci. Par la mesure des grandeurs physiques liées à un système durant ses activités, il est possible d'obtenir des informations au sujet d'un module ou d'un processus ciblé. Cette analyse, poussée à sont paroxysme, est combinée à des méthodes de traitements statistiques dans les attaques *side-channel*.

Analyse et extraction de caractéristiques. Cette dernière étape utilise les données recueillies dans les étapes précédentes pour caractériser le module que l'on souhaite étudier.

Pour illustrer l'imagerie, nous présentons le travail [139] dans lequel, à partir de l'analyse d'images prises avec un microscope électronique (SEM), l'auteur identifie les différentes portes logiques présentes dans un circuit intégré. La figure 1.6 montre comment, à partir d'images prises au niveau du silicium et au niveau de la couche métallique 1, il identifie une porte logique Non-Ou (NOR). Par la suite, la reconstruction de l'ensemble du schéma de câblage logique du circuit peut s'effectuer automatiquement à l'aide de l'informatique.

Nous illustrons le cas de l'analyse *side-channel* avec l'exemple d'une mesure d'émissions électromagnétiques d'un système durant l'exécution d'un processus de sécurité. Ceci permet, si aucune contremesure n'est implémentée, de déterminer les différentes phases d'exécution du processus dans le temps. De cette manière on obtient des informations pour élaborer une attaque plus sophistiquée sur ce mécanisme.



(a) Couche de Silicium (b) Niveau de Métal 1 (c) Identification des connexions

FIGURE 1.6 – Images issues d'un microscope SEM utilisé pour reconstruire une porte logique NOR (Source : [139])

1.2.2.2 Les attaques par injection de faute (FIA)

La génération de fautes contrôlées ou *Fault Injection* en anglais, est un ensemble de techniques qui a pour objectif de perturber volontairement le fonctionnement d'un système afin d'obtenir un comportement non prévu par ses concepteurs. Pour cela, un attaquant force le système ciblé à travailler en-dehors de ses conditions nominales de fonctionnement [144], *i.e.* en-dehors des conditions prévues par son cahier des charges. Ce contexte de fonctionnement anormal peut se cantonner à une partie du circuit ou en affecter la totalité, on parle respectivement de perturbations locales ou globales. Les fautes ainsi générées peuvent être temporaires, persistantes jusqu'au prochain redémarrage du système ou totalement irréversibles. Chacune d'entre elles peut être exploitée pour attaquer une sécurité

particulière. Nous allons présenter celles-ci.

1.2.2.2.1 Fautes sur les données Ce type de faute est observé lorsque la perturbation modifie une valeur dans le système ciblé. La modification de données peut avoir plusieurs utilisations. Par exemple, elle peut viser à corrompre un test de comparaison dans lequel des valeurs sont impliquées. Elle peut également servir comme complément d'information dans une attaque par analyse différentielle de fautes et ainsi permettre de retrouver la valeur d'une clef secrète. Cette dernière utilisation sera décrite dans la section 1.2.2.2.3. Une multitude d'autres applications est envisageable. La difficulté pour un attaquant réside cependant dans la précision et le contrôle de cette modification. Celle-ci peut dépendre de divers paramètres comme, par exemple, la localisation spatio-temporelle de l'endroit où injecter la perturbation. En fonction de cela, une donnée peut être changée de différentes manières :

Le bit. Dans ce cas, la modification apportée à la donnée ne se limite qu'à un seul bit. On observe les comportements suivant :

- Le *bit set* : la valeur d'un bit est montée à `1`.
- Le *bit reset* : la valeur d'un bit est baissée à `0`.
- Le *bit flip* : la valeur d'un bit est inversée. Autrement dit, `0` $\rightarrow$ `1` et `1` $\rightarrow$ `0`.

Un attaquant ayant une telle exactitude dans le contrôle du processus d'injection de fautes est un attaquant fort. Un des chemins d'attaques le plus utilisé pour avoir une telle précision est le laser. On peut également modifier un bit à l'aide de perturbations sur la tension d'alimentation ou par ondes électromagnétiques (EM) mais avec un contrôle moins précis.

Le mot. Avec une précision moindre, cette modification impacte un mot qui est considéré comme la plus petite taille de donnée manipulable par un processeur. Cette modification est spécifique à l'architecture du processeur. Pour une architecture n bits, les mots w auront une longueur de n bits ($n \in \mathbb{N}^*$). Ou plus formellement, soit w_n un mot de n bits tel que $w_n = \{b_i\}_{0 \leq i \leq n-1}$ et $b_i \in \mathbb{B} = \{0, 1\}$. Plusieurs changements sont envisageables :

- Valeur forcée : Le mot tout entier ou seulement une partie des bits est forcée à une valeur particulière. Par exemple pour une architecture 32 bit, le mot w_{32} aura les modifications suivantes : $w_{32} = \{1\}_{p \leq i \leq p+q}$, $w_{32} = \{0\}_{p \leq i \leq p+q}$ ou $w_{32} = \{\overline{b_i}\}_{p \leq i \leq p+q}$ avec $(p, q) \in \mathbb{N}^2$ et $0 \leq p + q \leq 31$.
- Valeur aléatoire : Une partie ou l'intégralité des bits d'un mot sont modifiés indépendamment les uns des autres provoquant la modification du mot w_n en w'_n.

1.2.2.2.2 Fautes sur flux d'exécution. Les fautes sur le flux d'exécution sont causées par la corruption d'instruction ou de test conditionnel. Toutefois, elles peuvent aussi être la conséquence de fautes sur des donnés décrites précédemment lorsque celles-ci s'appliquent sur une adresse.

Modification d'instruction. La modification d'instruction implique le remplacement de l'instruction originale en une seconde instruction qui modifie le comportement du système. Certaines des instructions remplacées sont ininterprétables par le CPU et soulèvent l'interruption *undefined instruction* sur les processeurs ARM. D'autres modifications permettent des comportements dangereux vis-à-vis de la sécurité. Par exemple, le travail [20] étudie les effets d'injection de faute sur l'exécution d'un programme. Au moyen de différents *glitches* sur le signal d'horloge, ils montrent la faisabilité et les applications envisageables de la modifications d'instructions.

Saut d'instruction. Le saut d'instruction permet de contourner des portions du code gênantes pour un attaquant. Diverses possibilités s'offrent ainsi. Par exemple, dans leurs travail [124], J.-M. Schmidt et C. Herbest expliquent comment ils attaquent l'algorithme de chiffrement RSA par saut d'instruction. A l'aide de *glitches* sur l'alimentation, ils contournent l'opération d'exponentiation rapide. Puisque cette opération est exécutée à chaque étape de l'algorithme, effectuer un saut d'instruction avant celle-ci altère le résultat final qui dépend de l'exposant secret. La modification fournit des informations à propos du secret.

En ce qui concerne les attaques de SoC par saut d'instruction, le travail de Timmers *et al.* [140] est un exemple remarquable. En effet, dans celui-ci, les auteurs contrôlent la valeur chargée dans le pointeur d'instructions en modifiant la valeur du registre ciblé dans une instruction de chargement mémoire. Nous expliquerons en détail cette manipulation qui vise le *Secure Boot* dans le chapitre 4 qui lui est dédié.

Test conditionnel. Le dernier type de modification du flux d'exécution concerne les tests logiques. La vérification d'une condition pour en exécuter une autre est un point critique de la sécurité. Corrompre ces tests peut s'effectuer selon différentes manières. La première méthode consiste en la modification des données à comparer. Ce cas rejoint les fautes sur les données décrites dans le paragraphe 1.2.2.2.1. La seconde méthode de corruption consiste en la modification du test lui-même. Par exemple, transformer un opérateur logique en son opposé, inverse le résultat de la comparaison. Perturber le test lui-même peut également amener ce dernier à valider un résultat sans se soucier des valeurs concernées. La perturbation de test conditionnel a été introduite par Barbu *et al.* dans [22]. Dans ce travail, ils arrivent à contourner le test d'autorisation de chargement d'un applet Java. La perturbation est obtenue par injection laser qui autorise le chargement d'un code malicieux. Cela peut également s'appliquer sur le protocole d'identification des cartes à puce à l'aide d'un code PIN. Perturber le processus qui contrôle la validité du code permet de contourner l'identification et d'utiliser la carte de manière non autorisée.

1.2.2.2.3 La Differential Fault Analysis (DFA). La cryptanalyses par perturbation est apparue après la publication de l'article [32]. La méthode proposée par les auteurs vise le protocole cryptographique RSA. Le principe de leur attaque est d'insérer une valeur erronée dans la multiplication des nombres premiers composant la clé. Si p et q sont premiers, factoriser $p \times q$ est très difficile. Tandis qu'en modifiant la valeur $p \rightarrow p'$, avec p' non premier, on retrouve aisément la valeur de q dans le produit $p' \times q$. Peu de temps

après, la faisabilité de ces attaques sur des algorithmes cryptographiques symétriques a été démontrée dans [30]. Les auteurs y présentent une attaque par analyse différentielle des fautes (DFA). En comparant deux à deux des chiffrés corrects avec des chiffrés erronés obtenus à partir d'un même texte clair, ils ont été capables de retrouver la valeur de la clef secrète d'un algorithme DES. D'autres améliorations ont permis d'appliquer des DFA sur des AES [60, 108]

La difficulté d'injecter des fautes de manière précise dans les SoC fait que l'étude de ce type d'attaque n'est pas très répandue dans la littérature scientifique. La complexité du paramétrage des attaques sur ce genre de système et la confidentialité des solutions technologiques présentes dans le commerce font que peu d'études publiques sont disponibles.

1.2.2.3 Les attaques par canaux auxiliaires (SCA)

Aussi connues sous leur appellation anglaise *side-channel attacks* (littéralement, attaques par canaux cachés) elles ont été introduites par le travail de Kocher [78]. Elles sont destinées à casser un processus algorithmique en s'appuyant sur des méthodes d'observation du matériel électronique qui l'exécute. Ces attaques visent particulièrement les implémentations d'algorithmes cryptographiques manipulant des données sensibles.

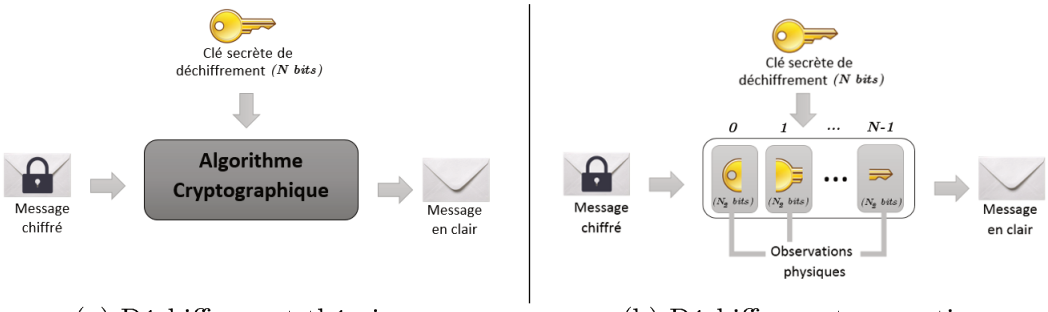
La théorie suppose que ces algorithmes sont suffisamment robustes pour protéger une donnée en clair. Le seul moyen de connaître celle-ci est de trouver la clé secrète qui la protège en essayant toutes les combinaisons possibles. Autrement dit, pour retrouver la valeur de la clé de N bits $K = \{K_0, K_1, \ldots, K_{N-1}\}$, il faut tester au maximum 2^N combinaisons de clés sur un couple clair/chiffré. Ce procédé est dénommé par l'expression « méthode par force brute ». Une donnée sera considérée comme suffisamment protégée si le temps que passe un attaquant à essayer les 2^N combinaisons de clés possibles est largement supérieur à la durée de vie du produit. Le nombre de bits N est donc crucial. Aujourd'hui par exemple, pour un chiffrement normalisé AES, la taille de clé minimum considérée comme suffisamment résistante est de $N = 128$ bits [3]. Cela sous-entend que si nous disposions d'un super ordinateur capable de tester mille milliards de valeurs clés par seconde (128×10^{12} $valeurs/s$), il faudrait un peu plus de la durée de vie de l'univers connu pour pouvoir toutes les tester.

Cependant, cette supposition ne tient pas compte de l'aspect physique du matériel. En effet, les algorithmes cryptographiques sont une succession d'opérations plus basiques appelées primitives cryptographiques. Les données sont donc manipulées de manière séquentielle à travers une succession de cycles et en fonction de la valeur de celles-ci, on observe une variation des grandeurs physiques du système. Pour schématiser, un '`1`' logique sera exprimé par une tension différente de celle représentant un '`0`' et la technologie utilisée pour la fabrication des circuits intégrés se comportera différemment suivant ces valeurs. Pour la majeur partie des circuits, la logique est réalisée avec des transistors de technologie CMOS qui offrent divers avantages au niveau intégration et consommation. Cependant, ces transistors ont des consommations de puissances électriques différentes selon les étapes de fonctionnement :

- Une consommation statique qui intervient lorsque les transistors sont bloqués. Elle est due aux courants de fuites et à la valeur de la tension de seuil [36].

- Une consommation dynamique qui est effective lors de la transition bloqué-passant ou passant-bloqué d'un transistor.

Ainsi, la variation des états physiques des processeurs qui effectuent différentes opérations cryptographiques est liée aux valeurs du message et de la clé. En s'appuyant sur cette propriété, les attaques par canaux auxiliaires ont montré un moyen de contourner la complexité $2^{\mathcal{O}(n)}$ des algorithmes cryptographiques. En effet, en mesurant la variation des grandeurs physiques telles que la consommation ou le temps d'exécution durant un calcul cryptographique, les attaques *side-channel* obtiennent des informations non prises en compte dans l'estimation théorique de la robustesse d'un algorithme. Comme l'illustre l'exemple de la figure 1.7 avec les mesures physiques, un attaquant peut identifier des zones d'exécution d'un algorithme dans lesquelles une grandeur physique dépend de la valeur d'un bit de clé. De cette manière, il peut travailler successivement sur les N bits. Ceci donne le double avantage d'avoir une information quant à la valeur du bit et de transformer la complexité exponentielle de la tâche $2^{\mathcal{O}(n)}$ en complexité linéaire $\mathcal{O}(n)$. De cette façon, le temps pour retrouver la clé est devenu largement inférieur à la durée de vie de la donnée à protéger.



(a) Déchiffrement théorique (b) Déchiffrement en pratique

FIGURE 1.7 – Informations supplémentaires obtenues par mesures physiques sur les systèmes effectuant le déchiffrement

Les attaques par canaux cachés sont toujours aussi redoutées dans le secteur de la sécurité numérique. Au fur et à mesure de l'évolution des technologies, différentes méthodes ont étés mises en place pour extraire une information en mesurant la variation de diverses grandeurs physiques de systèmes cryptographiques.

Les attaques SCA les plus couramment appliquées aujourd'hui sur des systèmes embarqués sont rapplées dans l'Annexe B.

Nécessité de traitement des mesures. Pour l'ensemble des attaques énumérées précédemment, il est nécessaire au préalable d'augmenter la qualité des mesures et de restreindre la taille des échantillons, d'une part, pour l'efficacité des attaques et d'autre part, pour diminuer la quantité de calculs. Ainsi, plusieurs mesures sont faites en se focalisant sur les zones contenant de l'information : les points d'intérêts (PoI). Ensuite, ces dernières sont traitées par des techniques du traitement du signal (moyenne, filtrage, alignement, etc.)

afin d'améliorer le rapport signal à bruit (RSB) des échantillons. Nous discutons ce point dans le chapitre 3 consacré à une attaque de module cryptographique.

1.2.3 Les attaques combinées (matérielle-logicielle)

Les attaques combinées sont une catégorie d'attaques qui associent des attaques matérielles à des attaques logicielles. Elles ont étés introduites en 2010 dans le travail de Barbu *et al.* [23] où les auteurs présentent un nouveau type d'attaque qui combine l'injection de faute et l'exploitation logicielle. Cette notion peut être étendue selon l'objectif de l'attaque et plusieurs cas de figures sont envisageables. Nous donnons quelques exemples :

- Une perturbation peut être mise en place pour contourner un contrôle de sécurité liée au chargement d'un logiciel malveillant. C'est le cas présenté dans [23], dans lequel une *applet* malicieuse est chargée dans un OS JAVA Card s'exécutant sur une carte à puce. Grâce à la perturbation du mécanisme de sécurité, non seulement les auteurs parviennent à charger un programme malicieux mais ils réussissent également à l'exécuter. Dans le contexte des SoC, la difficulté de la mise en place des injections de faute fait que le nombre d'études publiées à ce sujet est limité. La complexité de l'attaque demande une lourde préparation. Néanmoins, certains travaux remarquables utilisant cette méthode ont été réalisés. C'est le cas de [140] dans lequel les auteurs chargent et exécutent un programme malicieux en perturbant la copie d'un code depuis la mémoire flash vers la mémoire RAM à l'aide de *glitches*. La difficulté réside dans le contrôle et la précision de la faute injectée.

- Un logiciel peut être utilisé pour aider à la mise en place d'attaque matérielle. L'exécution d'un logiciel permettant de délimiter dans le temps un processus ciblé pour une attaque par injection de faute ou une attaque par canal auxiliaire peut être considéré comme une attaque combinée. Le logiciel peut, par exemple, générer des interruptions identifiables par analyse *side-channel* qui favorisent la détection du processus ciblé. Ainsi dans le cadre d'une attaque par fautes, cela permet de synchroniser la génération de la perturbation physique avec le processus ciblé et dans le cadre d'une attaque par canal auxiliaire, cela permet de localiser le motif recherché dans les signaux mesurés.

- Un logiciel conçu pour augmenter les fuites d'informations. Un logiciel peut être implémenté pour solliciter l'utilisation de données sensibles et ainsi permettre de les attaquer par canal auxiliaire. Prenons l'exemple d'une clef de chiffrement stockée de manière sécurisée, qui peut simplement être invoquée par un logiciel pour chiffrer des données qu'il souhaiterait sauvegarder de manière permanente. Le logiciel implémenté pour l'attaque, solliciterait cette clef de manière répétitive avec des données adaptées afin de pouvoir effectuer une attaque par canal auxiliaire.

De manière plus générale, les systèmes embarqués devenant de plus en plus complexes demandent l'élaboration d'attaques, elles aussi plus complexes. Les attaques combinées illustrent les nouvelles méthodes mises en place pour contourner les restrictions des protections de ces systèmes. La plupart des protections sont conçues pour parer un type de menace. Dans l'exemple de [23], il s'agissait de la vérification du chargement d'un code

certifié. Cependant cette sécurité n'est elle même pas protégée face aux perturbations. Par conséquent, en combinant deux chemins d'attaques simultanément (la perturbation et le chargement de code non vérifié), il est envisageable d'outrepasser la sécurité. La combinaison des chemins d'attaques a ainsi ouvert de nouvelles brèches dans la sécurité des systèmes. Une série d'attaques combinées a été mise en place dans le cadre de cette étude dans les chapitres 4 et 5. Ce sera l'occasion de décrire plus particulièrement en quoi elles peuvent consister sur un système sur puce.

1.2.4 Les chemins d'attaques matérielles

Les systèmes sur puces sont des systèmes électroniques composés de diverses ressources matérielles nécessitant des grandeurs physiques pour fonctionner. Déjà sur les microcontrôleurs, ces grandeurs étaient vues comme autant de chemins d'attaques. Les plus utilisées d'entre elles sont listées dans le tableau 1.1. Ce tableau introduit également un nouveau chemin d'attaque émergeant avec les systèmes complexes embarqués : le débogage matériel.

	Utilisation dans les attaques matérielles...	
Chemin d'attaque	**...par canal auxiliaire**	**...par injection de faute**
Alimentation (§1.2.4.1)	Analyse de la consommation électrique	Perturbation de l'alimentation
Rayonnement électromagnétique (§1.2.4.2)	Analyse des émissions électromagnétiques	Génération d'impulsions ou de signaux harmoniques
Temps d'exécution ou fréquence d'horloge (§1.2.4.3)	Mesure du temps d'exécution ou des fréquences de fonctionnement	Perturbation du signal d'horloge ou violations temporelles
Lumière (§1.2.4.4)	Analyse des photo-émissions	Perturbation par effet photo-électrique
Température (§1.2.4.5)	Analyse de la température du circuit	Fonctionnement du circuit en dehors de ses plages normales de températures

Nouveau chemin d'attaque	**Utilisation dans les attaques matérielles**
Débogage matériel (§1.2.4.6)	Largement utilisé dans les systèmes complexes, le débogage matériel peut être détourné pour accéder à des données sensibles du système

Tableau 1.1 – Principales grandeurs physiques potentiellement exploitables comme chemin d'attaque sur les systèmes complexes embarqués. (§x.x.x.x) désigne les sections qui décrivent les chemins d'attaque en question.

Etant donné la richesse de l'état de l'art concernant l'exploitation de ces grandeurs sur les microcontroleurs, nous avons envisagé de commencer à exploiter ces mêmes chemins d'attaques. Mais, avant toute chose, que ce soit pour les attaques matérielles ou combinées, nous allons définir trois niveaux d'intrusivités :

Définition 2. – **Invasif :** *un chemin d'attaque est dit invasif lorsqu'il est nécessaire de modifier de manière irréversible un circuit pour pouvoir l'attaquer. Cette modification est destructive, elle perturbe le fonctionnement normal de ce circuit et peut éventuellement le mettre hors-service.*

Définition 3. – **Semi-invasif :** *un chemin d'attaque est dit semi-invasif lorsqu'il consiste à ne modifier qu'une partie du système. La modification est irréversible mais laisse le circuit fonctionnel.*

Définition 4. – **Non invasif :** *un chemin d'attaque est non invasif lorsqu'il n'affecte pas l'intégrité physique du circuit attaqué. Ce dernier reste totalement fonctionnel.*

Pour illustrer ces définitions, on peut noter que la majorité des chemins d'attaques invasifs sont effectués dans le cadre de manipulations de rétro-ingénierie [142]. Les autres chemins sont plutôt destinés aux SCA et aux FIA.

De plus, nous allons spécifier deux notions de localité qui expriment la portée d'une attaque sur les zones du système. Ces notions sont valables aussi bien pour les attaques par canaux auxiliaires que pour les attaques par injection de fautes.

Définition 5. – **Globale :** *on dit qu'une attaque est globale lorsque l'action produite par le chemin utilisé concerne l'ensemble du système.*

Définition 6. – **Locale :** *à l'opposé, une attaque sera considérée comme locale, lorsque l'action est produite sur la fonctionnalité attaquée et n'affecte pas le reste du système.*

Maintenant, nous allons présenter les différents chemins d'attaque listés dans le tableau 1.1. Pour chaque paragraphe décrivant un chemin d'attaque, la partie précédée du symbole « ▶ » aborde le type de difficultés que l'ont peut rencontrer lorsque l'on exploite ce dernier pour un SoC.

1.2.4.1 Les alimentations électriques

1.2.4.1.1 L'alimentation électrique pour les SCA. Historiquement, il s'agit de la première grandeur physique exploitée dans les attaques matérielles. Pour mesurer l'alimentation d'un circuit, deux méthodes sont utilisées : la mesure de l'alimentation principale et la mesure par *probing*.

Mesure de la puissance de l'alimentation. La mesure de la consommation en puissance d'un circuit est un moyen détourné pour observer son comportement [88]. Cette idée a été initialement présentée dans le travail de P. Kocher pour effectuer une attaque sur un microcontrôleur sécurisé [79]. Dans celui-ci, en appliquant l'hypothèse de relation entre la consommation globale de courant et les opérations effectuées par un système électronique, Kocher extrait, entre autres, des clefs secrètes de chiffrements. Pour mesurer cette consommation, il intercale une faible résistance (ex. 1 Ω) entre le microcontrôleur et sa masse. La

tension mesurée à ses bornes, divisée par la valeur de la résistance, représente le courant que consomme le circuit. La mesure de cette tension fournit de précieuses informations quant aux processus exécutés et nécessite une potentielle modification du circuit ciblé.

Mesures par probing. Un moyen d'observation de la consommation électrique plus localisé a été proposé dans [64]. Pour effectuer la mesure, les auteurs proposent d'utiliser de fines aiguilles dont la taille est de l'ordre du dixième de μm pour observer les consommations électriques dans les pistes du circuit. Il s'agit d'une attaque semi-invasive qui permet de mesurer un signal donné. De nos jours, il existe des stations de *probing* adaptées proposant des pointes de différentes tailles et des mécanismes de positionnement précis pour optimiser une mesure.

► **Difficultés liées aux SoC.** Si la mesure de la consommation de puissance est envisageable sur un microcontrôleur, elle est beaucoup plus compliquée à mettre en place sur des systèmes polyvalents tels que les SoC. Sans prendre en compte les protections implémentées depuis la publication, entre autres, des attaques de Kocher, la difficulté provient tout d'abord de la complexité des architectures. Comme il est rappelé dans l'annexe A de ce document, les systèmes sur puces sont composés d'une multitude de sous-systèmes. La mesure de l'alimentation d'un module en particulier représente un réel challenge. En effet, il n'y a pas une alimentation pour tout le circuit, mais plusieurs pour les différents sous-systèmes. L'alternative d'effectuer une mesure par *probing* est d'autant plus compliquée que la technologie de gravure du SoC est fine (voir Annexe A.2). Une autre difficulté est ajoutée par la gestion dynamique de la consommation (DVFS), qui optimise la consommation du circuit en fonction des besoins. Cette gestion crée des perturbations sur les signaux et implique une augmentation du nombre de mesures [100]. Pour finir, les diverses capacités de découplage montées sur l'alimentation ne permettent pas d'effectuer des mesures de manière non invasive. Afin de pouvoir mesurer les variations de tension, il faut dessouder ces capacités et donc dégrader la stabilité de l'alimentation du SoC ce qui entraîne son mauvais fonctionnement.

Par conséquent, mesurer la tension d'un contrôleur, en particulier au sein d'un système multiprocesseurs composé d'une multitude de périphériques autonomes, possédant plusieurs alimentations sur plusieurs niveaux de métal, requiert plus d'efforts que pour un simple microcontrôleur. Pour plus de détails au sujet des systèmes de gestion de puissance et des horloges dans les SoC, se reporter à l'Annexe A.4 de ce document.

1.2.4.1.2 L'alimentation électrique pour les FIA. De la même manière que pour la mesure de la consommation électrique, les perturbations électriques peuvent s'effectuer de manière globale ou locale. Globale en agissant sur l'alimentation principale, locale en utilisant une fine aiguille.

La variation de l'alimentation principale. Une brusque chute temporaire (*under powering* [125]) ou un pic (*glitch* [75]) sur la tension d'alimentation d'un système durant le déroulement d'un processus peut provoquer des erreurs d'exécution, qui sont en général de mauvaises interprétations ou le saut d'une ou plusieurs instructions. Il s'agit là d'une per-

turbation globale, fonction de la durée et de l'intensité de la brusque variation de tension, autrement dit cette perturbation affecte l'ensemble du système.

L'injection de fautes par probing. La technique du *probing* décrite dans le paragraphe 1.2.4.1.1 est également utilisée pour perturber un signal donné. En plaçant de manière semi-invasive une aiguille sur la piste de signal à perturber, un attaquant peut induire localement un *glitch* dans le système.

▶ **Difficultés liées aux SoC.** Les difficultés pour injecter une faute sur l'alimentation des systèmes complexes sont similaires à celles rencontrées pour mesurer la tension d'alimentation. En partant du fait que les tensions d'alimentation de la logique sont générées en interne, lorsque l'on souhaite générer un *glitch* sur l'alimentation d'un module particulier au sein d'un SoC, les principales difficultés sont :

- La présence de multiples arbres d'alimentation. Identifier celui qui alimente le module que l'on souhaite attaquer demande une certaine analyse préalable du système. De plus, perturber un arbre affecte l'ensemble des modules qu'il alimente. Ceci risque générer des perturbations non voulues.

- La difficulté d'accès aux alimentations. En fonction du circuit imprimé sur lequel est monté le SoC et le type de boîtier de celui-ci, il arrive que l'alimentation ciblée ne soit pas accessible sans devoir dessouder des composants.

- Comme pour la mesure de l'alimentation, la présence des capacités de découplage peut limiter l'effet de la perturbation. Les condensateurs placés sur les alimentations auront tendance à lisser les *glitches* générés pour les attaques.

Les Body-Bias Injection. Le *Body-Bias Injection* (BBI) est une méthode qui exploite le *body effect* (voir annexe A.4.1). A l'instar de la technique du *probing*, la BBI utilise elle aussi une fine aiguille pour injecter des erreurs non plus sur les pistes, mais directement dans le substrat du circuit. L'attaque consiste à placer l'aiguille et à injecter une impulsion électrique positive ou négative dans une zone déterminée du substrat. Il s'agit d'une attaque semi-invasive puisque pour accéder au substrat, la puce doit être préalablement mise à nu. Ce type de perturbation est locale, elle affecte une zone restreinte du circuit. Comme pour les *glitch*, la perturbation générée est fonction de l'amplitude, de la durée et de la polarité de l'impulsion émise [91, 28, 141].

▶ **Difficultés liées aux SoC.** La BBI n'est possible que s'il est possible d'accéder à la partie active de la puce afin de pouvoir mettre en contact l'aiguille métallique avec le substrat. Pour cela il est nécessaire de préparer de manière semi-invasive le SoC. De plus, en fonction de la finesse de gravure utilisée pour réaliser le SoC (voir annexe A.2), la taille de l'aiguille de quelques dixièmes de μm, a plus ou moins d'incidence.

1.2.4.2 Les rayonnements électromagnétiques (EM)

1.2.4.2.1 Les rayonnements électromagnétiques pour les SCA. Suite aux études sur la consommation électrique, un important travail de recherches a été effectué pour mettre en évidence les relations entre les activités d'un circuit, et le rayonnement électromagnétique.

Mesures des rayonnements électromagnétiques. La mesure des émissions EM pour attaquer un circuit en fonctionnement a été proposée par les Security Labs de l'entreprise Gemplus dans [57]. L'attaque exploite l'effet d'induction des champs électromagnétiques pour extraire des informations sensibles liées aux processus exécutés. L'hypothèse ici est que, d'une part, la variation des courants internes aux circuits est fonction des opérations effectuées et, d'autre part, que ces courants induisent des champs magnétiques $\overrightarrow{\mathcal{H}}$ eux aussi liés aux opérations effectuées. Les mesures sont réalisées avec des sondes constituées d'un corps en ferrite et d'un bobinage cuivré. Un des avantages de cette mesure par rapport à celle du courant, est la possibilité d'effectuer des mesures locales sans forcément être au contact avec la puce. En effet, les champs EM ont la propriété de traverser certains matériaux et la mesure locale s'obtient en plaçant la sonde au-dessus de la partie ciblée du circuit. Les champs EM sont des grandeurs physiques dépendantes du temps sur 3 dimensions spatiales qui offrent plus de possibilités que la simple mesure de la consommation de puissance. En revanche, ces nouveaux degrés de liberté représentent également de nouveaux paramètres à configurer pour procéder aux mesures. Le choix du type de sonde et son orientation sont des paramètres peu évidents à définir à priori. De plus, la localisation des zones de la puce émettant des informations pertinentes demande un travail supplémentaire.

Néanmoins, les champs électromagnétiques sont très couramment utilisés au sein de la communauté de la sécurité numérique et ont étés déclinés selon une multitude d'attaques et contre-mesures comme le décrit le travail [46].

► **Difficultés liées aux SoC.** Le rayonnement EM est probablement la grandeur physique la plus utile pour les attaques matérielles sur les SoC. En effet, les champs EM sont une bonne alternative si on considère les difficultés évoquées lors de la mesure de la puissance de l'alimentation section 1.2.4.1.

Cependant, de nouvelles contraintes sont à prendre en compte. Premièrement, la distance. La loi de Biot et Savart énonce que l'intensité du champs magnétique décroît proportionnellement à l'inverse du carré de la distance. Donc, pour mesurer des signaux exploitables il est nécessaire de se placer au plus près du système ciblé. Deuxièmement, certains matériaux empêchent la propagation des ondes EM. Par conséquent, le type du boîtier et le contexte dans lequel le SoC se situe, déterminent la faisabilité d'une telle analyse. Par exemple, certains boîtiers font office de protections comme les radiateurs métalliques dissipateurs de chaleur qui bloquent ou réduisent les rayonnements vers l'extérieur (voir Annexe A.1). Plus généralement, tout conditionnement qui atténue trop fortement les signaux est un obstacle qui rend impossible l'analyse par mesure de champs EM sans modifications de la cible.

1.2.4.2.2 Les rayonnements électromagnétiques pour les Attaque FIA. Les champs électromagnétiques sont également exploités pour injecter des fautes dans les circuits en générant des courants par effet d'induction. Deux sortes d'ondes électromagnétiques sont

utilisées : les impulsions et les sinusoïdes.

Les impulsions EM. L'utilisation des champs magnétiques pour perturber un système informatique volontairement a été introduite dans [120]. Cette technique utilise le phénomène d'induction électromagnétique. Pour injecter des fautes, le principe est de générer localement des courants induits au sein d'un composant grâce à de puissantes impulsions EM ajustables en intensité et en durée. Ce type de perturbation a été utilisé dans [47] pour produire des fautes dans un microcontrôleur ainsi que dans [98] pour évaluer la susceptibilité EM des circuits.

Les harmoniques EM. La perturbation du système ciblé se fait en générant à proximité des ondes électromagnétiques sinusoïdales dont les amplitudes et les fréquences peuvent être paramétrées. Ces types de signaux ont été utilisés dans [25] pour biaiser un générateur de nombres aléatoires *True Random Number Generator* (TRNG).

► **Difficultés liées aux SoC.** Que ce soit pour la technique d'injection de faute par impulsions ou par harmoniques, on retrouve ici les difficultés liées à l'utilisation des ondes électromagnétiques mentionnées dans la partie consacrée aux mesures de leur rayonnement.

Une première partie d'entre elles concerne la possibilité d'utiliser cette grandeur physique pour perturber les SoC. Les matériaux du boîtier ou un agencement de circuit imprimé particulier risquent de compromettre la manipulation. En effet, il faut que le champ puisse traverser le boîtier et que son intensité soit suffisante afin de créer des perturbations. Autrement dit, un boîtier en métal ou trop épais ou un agencement de circuit imprimé particulier interdit l'utilisation des ondes EM pour injecter des fautes sans entreprendre des modifications invasives parfois compliquées.

L'autre partie des difficultés à considérer concerne l'ajustement de tous les paramètres que comporte cette grandeur physique. Les SoC ont des surfaces de silicium supérieures à celle des microcontrôleurs. Le travail de recherche de la zone à perturber est bien plus conséquent. Déjà pour un unique point spatial donné, si on considère tous les paramètres à ajuster pour l'injection de perturbation :

- par impulsions EM : intensité des impulsions, leur polarité, leur durée et les instants auxquels les déclencher.

- par harmoniques EM : amplitude des sinusoïdes, leur polarité, leur fréquence et leur phase.

Cela représente un travail d'optimisation non négligeable. Mais si cet ajustement doit s'étendre à l'ensemble des points de la surface d'un SoC, le travail est d'autant plus long. Un autre point concerne le choix déterminant des sondes d'injection. Le champ généré par une sonde est fonction de sa taille, de la forme de la ferrite, de la taille du fil, du nombre de spires, etc. Tous ces paramètres influencent les effets produits sur le système à perturber [101, 98].

1.2.4.3 Le temps

1.2.4.3.1 Le temps pour les SCA. Le fonctionnement des circuits est cadencé par des signaux d'horloge. Par conséquent l'exécution d'un processus laisse une signature temporelle.

Les timing attacks. Le traitement des données et les branches conditionnelles influencent la durée d'exécution d'un algorithme. Cette constatation a été exploitée pour la première fois par Kocher dans [78] où il présente la *timing attack* (attaque temporelle en français). Le principe de cette attaque est d'identifier dans l'algorithme des opérations ayant une signature temporelle significative et dépendante de la valeur des bits du secret. Dans [78], l'attaque porte sur exponentiation modulaire dans laquelle est effectuée une multiplication seulement si la valeur du bit de la clef secrète est "1". La multiplication étant une opération coûteuse en temps, on dispose ainsi d'un moyen de discriminer ces bits et retrouver la valeur du secret.

► **Difficultés liées aux SoC.** Depuis l'apparition des *timing attacks*, de nombreuses contremesures ont étés implémentées pour équilibrer la durée entre les différents calculs sensibles. Des opérations servant de leurres sont ajoutées pour que les temps mesurés soient indiscernables en fonction des valeurs manipulées.

1.2.4.3.2 Le temps pour les FIA.

Variation du signal d'horloge. La génération d'un *glitch* sur le signal d'horloge implique une violation du timing nécessaire au bon fonctionnement du circuit [151].

L'autre méthode pour perturber les signaux d'horloge est l'utilisation des harmoniques EM. Ce moyen est décrit dans [110] pour perturber un oscillateur lié au signal d'horloge.

► **Difficultés liées aux SoC.** Les systèmes sur la puce génèrent eux-mêmes leurs signaux d'horloge (voir Annexe A.4). Il faut utiliser une grandeur physique adaptée pour perturber ces derniers. Par exemple, la perturbation utilisant les champs EM est envisageable en considérant les contraintes liées à cette grandeur physique.

1.2.4.4 La lumière

1.2.4.4.1 La lumière pour les SCA.

La photo-émission. Un circuit intégré est composé d'une multitude de transistors, chacun représentant deux états logiques "0" ou "1". La transition d'un état à un autre produit l'émission de quelques photons. Le système *Picosecond imaging circuit analysis* [143], initialement développé pour détecter les transistors défaillants, permet de mesurer ces photons. Ce système de mesure a été utilisé dans les travaux de [123, 52] pour montrer que les photons étaient liés à la donnée manipulée. Ils illustrent leur travail en attaquant un algorithme de chiffrement AES.

► **Difficultés liées aux SoC.** La photo-émission n'émet que quelques photons par changement d'état, il est donc nécessaire de préparer le circuit de manière semi-invasive pour dégager les zones d'émission. Autrement dit, il faut enlever le boîtier pour pouvoir appliquer ce type d'attaque. De plus, l'utilisation de cette méthode dépend de l'orientation de la puce. Elle convient parfaitement lorsque la face active est directement accessible. En revanche, elle est difficilement applicable lorsque les couches métalliques successivement placées entre les transistors ciblés et le dispositif de mesure absorbent les photons émis. D'autre part, afin de capter l'ensemble des charges déposées par les photons sur les capteurs utilisés, ce type d'attaque nécessite de répéter la séquence attaquée plusieurs fois ce qui est parfois difficile lorsque le SoC est géré par un OS complexe.

1.2.4.4.2 La lumière pour les FIA.

La lumière blanche. L'effet photoélectrique rend les circuits sensibles à la lumière. Par conséquent, si un circuit est exposé un bref instant à un intense flash de lumière, les courants induits par les photons peuvent être utilisés comme moyens de perturbation. La génération d'un flash sur l'ensemble du circuit est une perturbation globale. Ce type d'attaque a été introduit en 2002 par Skorobogatov et Anderson [130].

Le laser. Les effets du laser sur les semi-conducteurs ont été mis en avant dans [62]. Les comportements produits sont semblables à ceux obtenus avec de la lumière blanche mais l'avantage du laser est la possibilité de viser précisément une zone du circuit [54, 111]. C'est donc une perturbation locale.

► **Difficultés liées aux SoC.** Lorsque cela est possible, autrement dit lorsque la face active du composant n'est pas sous la face arrière du SoC, une préparation physique est nécessaire pour que le laser puisse accéder aux portes logiques du système. Dans le cas contraire, le laser n'accèdera qu'aux couches métalliques. Quoi qu'il arrive, la grande précision du tir laser (de l'ordre du μm) fait de celui-ci un vecteur d'attaque privilégié si l'accès à la puce est possible.

Les rayons X. Pour des raisons de fiabilité en milieux hostiles, les concepteurs de circuits intégrés étudient déjà depuis des dizaines d'années les effets que peuvent avoir l'environnement externe sur leurs systèmes. Par exemple, pour des systèmes destinés à être embarqués dans l'espace, l'entreprise IBM a conduit une étude sur l'influence des fautes qui peuvent être induites par des rayonnements cosmiques [150]. Plus tard, l'idée d'utiliser ce moyen pour perturber volontairement un système sécurisé a été reprise par la communauté de la sécurité numérique et l'utilisation des rayons-X pour fauter un appareil électronique a été proposée dans divers travaux [21, 130]. Dernièrement dans [14], les auteurs ont montré la faisabilité d'utiliser un synchrotron afin d'obtenir un faisceau de rayons-X focalisé sur une dizaine de nanomètres. D'autres types de rayonnements pourraient également perturber le fonctionnement des circuits mais, que ce soit pour les rayons-X ou les autres rayonnements cosmiques, le matériel et l'expertise nécessaires pour la mise en œuvre d'une attaque précise restent aujourd'hui hors d'atteinte pour presque tous les attaquants.

1.2.4.5 La température.

1.2.4.5.1 La température dans les attaques SCA.

La thermographie infra-rouge. La thermographie infra-rouge (IR) met en avant les points chauds induits par le passage du courant. Ce courant génère un échauffement local qui se diffuse dans le composant. Ainsi la thermographie peut permettre de localiser spatialement des blocs fonctionnels [33].

► **Difficultés liées aux SoC.** La mesure par thermographie IR n'est efficace que s'il n'existe aucun obstacle entre l'appareil de mesure et la zone émettrice. La diffusion de chaleur dépend du matériau du boîtier, par conséquent un circuit intégré ayant un boîtier trop épais ou trop de couches métalliques limite les informations obtenues par cette méthode. Un retrait de l'ensemble ou d'une partie du boîtier sera donc nécessaire.

1.2.4.5.2 La température dans les FIA.

Température. Le fait de faire fonctionner un circuit en dehors des plages de températures pour lesquelles il a été conçu provoque des dysfonctionnements. L'augmentation de la température est une perturbation globale du système provoquant deux effets notables. Le premier est la perturbation des générateurs de nombres aléatoires [132]. Et le deuxième concerne la violation des contraintes temporelles dans le fonctionnement des circuits synchrones comme les mémoires non volatiles. Dans ces dernières, au même titre que la modification des paramètres électriques et du temps de propagation dans le circuit, la modification de la température au-delà d'un certain seuil, fait que les opérations de lecture et d'écriture s'effectuent ou pas et ce, de manière indépendante [21].

Cold boot attack. Le refroidissement intense des mémoires vives DRAM et SRAM permet d'exploiter l'effet de rémanence des données inhérentes à ces technologies [63, 40]. Après la coupure de l'alimentation, la durée de persistance des données au sein de ces mémoires peut être prolongée en réfrigérant fortement ces dernières. Ainsi, il est possible d'accéder aux données durant un certain laps de temps. Ce type de manipulation est utilisée, par exemple, pour subtiliser des clefs cryptographiques utilisées lors du démarrage des PC dont les disques durs ont été chiffrés. Elles sont connues sous le nom de *cold boot attack* [66].

► **Difficultés liées aux SoC.** Le refroidissement de la *cold boot attack* est envisageable pour exploiter les effets de rémanence des mémoires des SoC mais l'accès à ces dernières est plus difficile. En effet, les mémoires vives internes que l'on pourrait cibler ne sont pas aisément accessibles pour un attaquant. L'annexe A.3 rappelle qu'une partie d'entre elles est interne aux SoC et que l'autre est soudée sur le circuit imprimé accueillant le circuit intégré. Une modification du circuit doit être effectuée afin de pouvoir communiquer avec ces mémoires au bon moment.

1.2.4.6 Le débogage matériel

Les solutions de débogage matériel ont initialement été conçues pour observer les systèmes afin de corriger d'éventuelles erreurs. Elles permettent notamment d'accéder à leurs registres et à leurs mémoires afin de vérifier leur bon fonctionnement. Les deux protocoles les plus répandus aujourd'hui sont le JTAG et le SWD (voir Annexe D). Cependant, le large panel de fonctionnalités que proposent ces solutions représente une menace du point de vue de la sécurité des données. Ces systèmes de débogage peuvent êtres détournés pour être utilisés comme chemins d'attaque permettant de lire des données qui initialement n'étaient accessibles qu'aux concepteurs et aux développeurs [147, 119, 1]. De cette manière, le débogage matériel est utilisé comme canal auxiliaire pour dérober des informations sensibles. Ce nouveau chemin d'attaque très puissant est spécifique aux systèmes complexes embarqués. En effet, sur ces systèmes, les concepteurs et les fabricants doivent mettre en place des moyens de vérification performants pour localiser et corriger le moindre défaut. Pour cette raison, on trouve systématiquement du JTAG ou du SWD sur tous les SoC du marché. Cependant, ils représentent une vraie « porte dérobée » donnant accès sur l'ensemble du système. Le chapitre 5 de cette étude est consacré à l'étude du débogage matériel.

1.3 La sécurité des SoC

Un besoin de sécurité

L'ensemble des menaces présentées dans la section précédente plane sur les systèmes sur puces et oblige leurs concepteurs à les doter de systèmes de sécurité. Ainsi les SoC se voient équipés de différents modules consacrés à la sécurité qui assurent la confidentialité, l'intégrité et la fiabilité de leurs services et de leurs données. Cependant, aucun standard, aucune norme ne régit la cohérence de ces sécurités. La majorité de ces modules existaient déjà et leur adaptation dans les SoC est en partie due à un choix arbitraire des concepteurs [77]. Toujours avec le principe d'architecture modulaire et hétérogène, chacun de ces modules est destiné à assurer une fonction de sécurité. Par exemple, les cryptoprocesseurs chiffrent les données sensibles, les mémoires sécurisés protègent les accès des clés utilisées pour les chiffrements et les *watchdog* surveillent toute intrusion. Ces modules constituent une chaîne dont la solidité est égale à la résistance maximale de son maillon le plus faible. Autrement dit, la sécurité d'un SoC est un ensemble dont la solidité globale dépend de la solidité de l'élément le plus faible. Un attaquant brisant la sécurité d'un des modules peut par la suite exploiter ce dernier pour casser la sécurité des autres. Ceci rend d'autant plus difficile la caractérisation du niveau de protection atteint. Il faut être en mesure de déterminer quel est l'élément qui représente le talon d'Achille du système.

L'étude dans les SoC du comportement de ces systèmes vis-à-vis des attaques matérielles est l'enjeu de ce travail. Cette section donne une vue d'ensemble des modules matériels impliqués dans la sécurité. Elle décrit leurs objectifs et leurs principes de fonctionnement.

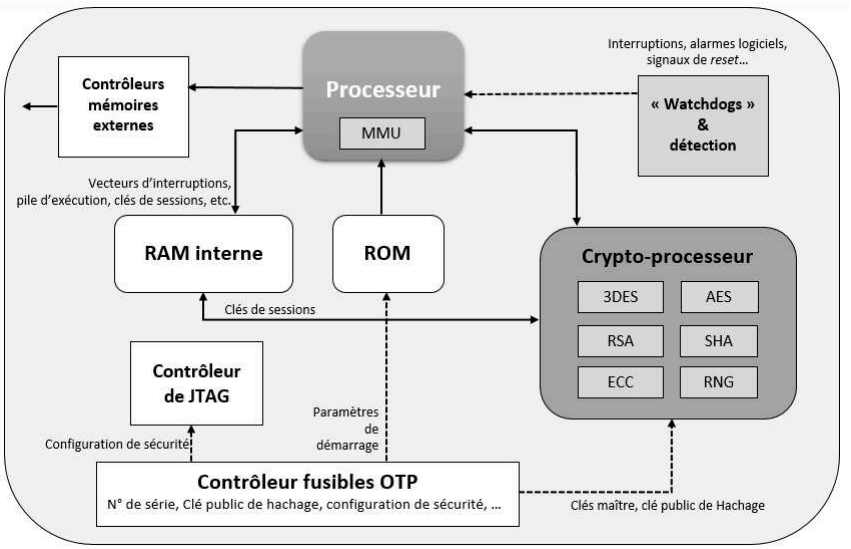


FIGURE 1.8 – Schéma simplifié des modules de sécurité présents dans les SoC

Sur l'ensemble des SoC du marché, on observe des similitudes entre les différentes solutions de sécurité adoptée par les concepteurs. La figure 1.8 présente les modules de sécurité les plus caractéristiques que l'on trouve pratiquement sur tous les SoC du marché actuel. A savoir :

1. Les unités de la gestion de la mémoire (MMU).
2. Les mémoires internes.
3. Les mémoires OTP et les fusibles.
4. Les coprocesseurs cryptographiques.
5. Les contrôleurs de débogage.
6. La détection.
7. L'isolation matérielle des processus.

Afin d'avoir une vue globale de ces systèmes, ces modules sont brièvement décrits dans les paragraphes suivants. Certaines caractéristiques des modules sur lesquels nos études se portent sont approfondies dans les chapitres correspondants.

1.3.1 Les unités de la gestion de la mémoire

Une unité de gestion de la mémoire, aussi connue sous le nom anglais *Memory Management Unit* (MMU), est une unité qui administre la cartographie et les privilèges d'accès des mémoires utilisées par un processeur selon des tables dites *Translation Lookaside Buffer* (TLB). Ces tables gèrent la correspondance entre les adresses virtuelles et les adresses physiques pour les différentes mémoires volatiles utilisées. La MMU contrôle également les permissions d'accès aux différentes pages de données. Cette dernière caractéristique sert aux systèmes d'exploitation tels que Linux pour isoler la partie noyau (*kernel*) de la partie utilisateurs (*user*). En effet, ces deux parties possèdent des niveaux de privilèges différents. Le noyau qui fait l'intermédiaire entre le matériel et l'espace utilisateur nécessite certains privilèges pour être manipulé. Un simple utilisateur exécute des processus dans son espace et demande l'accès au matériel grâce à l'interface du noyau. Ce dernier autorise ou non les requêtes en fonction des permissions attribuées à cet utilisateur. Ceci permet à l'OS de prévenir le fait qu'un processus utilisateur atteigne des zones mémoires auxquelles il ne devrait pas avoir accès. De la même manière, les parties mémoires allouées à différents processus sont isolées entre elles pour éviter la modification respective de leurs données. Une MMU est habituellement implémentée pour fonctionner directement avec les processeurs principaux d'un système et n'a aucune possibilité de contrôler les accès autorisés aux mémoires par les autres modules, ce qui inclut d'autres contrôleurs et les mécanismes *Direct Memory Access* (DMA) [44] (Chapitres "System Control" et "Memory Managment Unit").

1.3.2 Les mémoires internes

Tous les systèmes sur puces intègrent des mémoires ROM et des RAM internes ou *On-Chip RAM* (OCRAM). Ces mémoires sont de faible capacité (quelques centaines de *k*o). La ROM contient le premier code qui va être exécuté lors de l'allumage ou du redémarrage d'un SoC. La OCRAM a deux fonctions : dans un premier temps, elle sert lors du démarrage du système, puis elle peut être utilisée pour stocker certaines données critiques du système (vecteurs d'exceptions, pile d'exécution, etc.) [80].

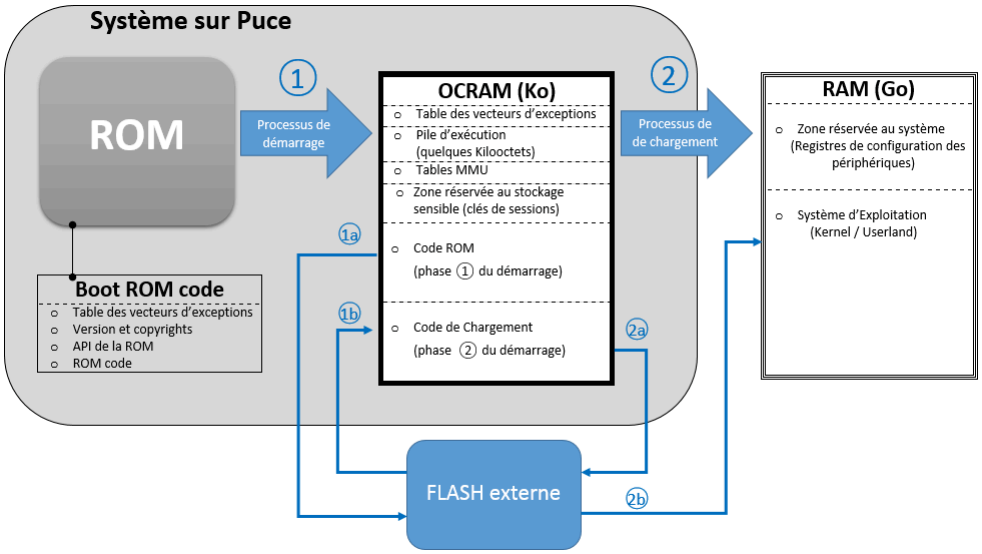


FIGURE 1.9 – Principe d'utilisation de la RAM interne dans la séquence de démarrage

La Figure 1.9 représente l'utilisation de ces mémoires lors du démarrage du système. Cette séquence est couramment dénommée la séquence de *boot*. Etant donné que le code ROM est immutable, il est nécessaire que les adresses ainsi que les caractéristiques de la mémoire RAM dans laquelle il chargera ses données, soient connues lors de son implémentation. C'est pourquoi, juste après la mise sous tension du système, la mémoire ROM utilise temporairement la RAM interne pour y copier son code et des données nécessaires à la séquence de démarrage.

En ce qui concerne le stockage de données critiques pour le système, il est également courant que les OCRAM conservent les tables d'adressage de la MMU et la pile du CPU. Les permissions pour l'accès aux différentes partitions de cette mémoire sont définies par la MMU. Les accès non sécurisés sont détectés par des interruptions. Dans ces mémoires on peut également trouver le stockage de diverses clefs cryptographiques utilisées par les périphériques : les clefs de sessions.

L'Annexe A.3 donne de plus amples détails à propos du rôle des différentes mémoires présentes dans les SoC.

1.3.3 Les mémoires OTP et les fusibles

Les mémoires OTP ou fusibles, sont la pierre angulaire de la sécurité des SoC. En effet, une partie de leurs valeurs définit le système (ex. le n° de série) alors qu'une autre partie est utilisée dès le démarrage pour paramétrer sa sécurité (accès au débogage, privilèges divers, clef maître cryptographique, etc.). Ces valeurs ne sont pas directement accessibles aux périphériques susceptibles d'en avoir besoin. C'est un contrôleur spécifique qui sert d'interface en utilisant le principe des *shadow registers*. Il s'agit de mémoires volatiles qui contiennent une copie des valeurs des fusibles. A chaque mise en marche du SoC et lorsque cela lui est demandé, le contrôleur copie les informations des fusibles vers cette mémoire. Ceci offre l'avantage d'avoir un accès rapide aux données et la possibilité de modifier ces dernières, par exemple pour la programmation des fusibles ou pour travailler en mode débogage. L'accès aux *shadow registers* se fait soit grâce aux DMA intégrés dans certains contrôleurs soit au travers de la mémoire principale dans laquelle ils sont cartographiés. Certains signaux spécifiques peuvent être directement câblés. C'est le cas par exemple pour le signal de configuration d'accès au débogage qui doit être pris en compte dans les tous premiers instants. Pour sécuriser l'accès aux OTP, certains d'entre eux ont pour fonction de configurer le contrôleur OTP pour qu'il fige la valeur des ses *shadow registers*. D'autres OTP ont pour fonction de rendre ces *shadow registers* uniquement lisibles à des processus ayant les privilèges adéquats.

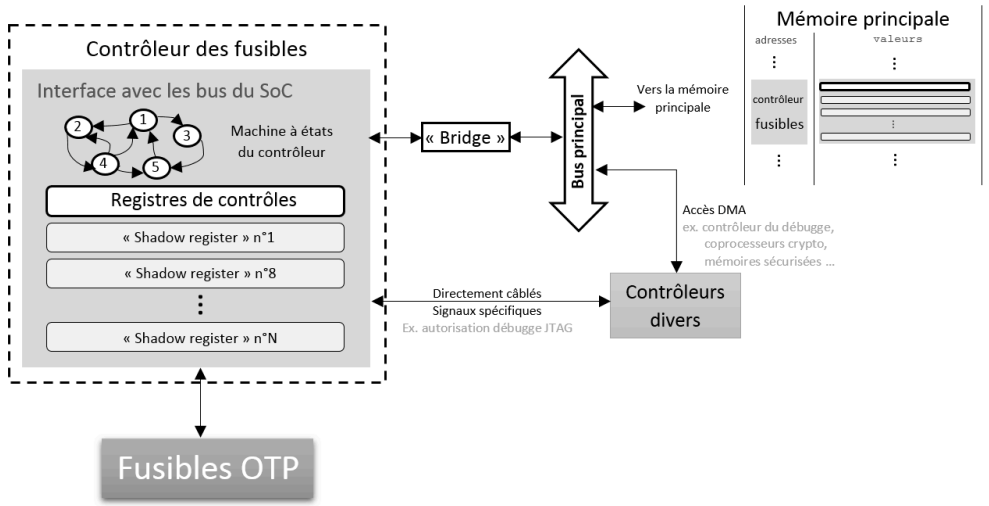


FIGURE 1.10 – Exemple de système de gestion des fusibles

La figure 1.10 illustre un exemple d'implémentation d'un contrôleur de fusible dans un SoC. Une description des OTP est également disponible dans l'Annexe A.3.1.2 consacrée aux différentes mémoires présentes dans les SoC.

1.3.4 Les coprocesseurs cryptographiques

Pour garantir la sécurité, la confidentialité, l'intégrité des données tout en conservant de bonnes performances, des coprocesseurs cryptographiques sont intégrés dans les SoC [51]. Il s'agit de processeurs travaillant en parallèles du processeur principal. Ils intègrent des accélérateurs cryptographiques matériels optimisés afin d'effectuer des calculs rapidement, avec une basse consommation [71] tout en résistant aux attaques matérielles [112]. Dans les SoC, pour des raisons de sécurité entre autres, ces processeurs ont une certaine autonomie vis-à-vis du CPU. Autrement dit, ils possèdent leur propre horloge, leurs zones mémoires voir leurs propres mémoires et un DMA associé. Il existe autant d'accélérateurs cryptographiques matériels qu'il y a d'algorithmes, même si certaines de leurs parties sont mutuellement utilisables [4]. Pour synthétiser, il est possible de classer ces accélérateurs suivant les catégories d'algorithmes cryptographiques qu'ils exécutent : symétriques, asymétriques et fonctions de hachages. Ils sont contrôlés depuis des banques de registres et lèvent généralement des interruptions pour signaler une information au système (fin de travail, erreur, détection...).

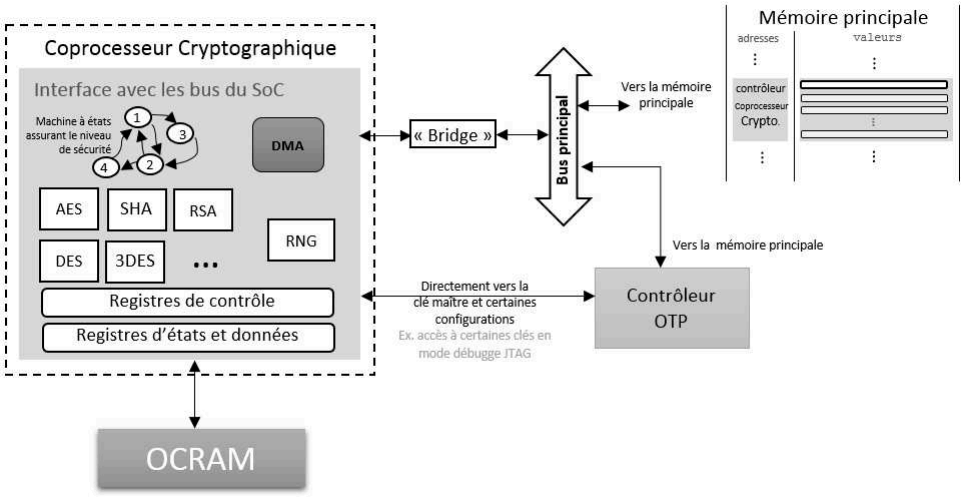


FIGURE 1.11 – Exemple de coprocesseur cryptographique intégré

La figure 1.11 présente une intégration typique de coprocesseur au sein d'un SoC. Les accélérateurs cryptographiques matériels ne sont pas accessibles directement. Pour les commander, un programme devra utiliser l'interface du coprocesseur constitué de registres de contrôle et de status. Une machine d'état informe de la disponibilité des ressources et des restrictions sont appliquées sur celles-ci en fonction du niveau de menace détecté par différents *watchdogs*. Dans cet exemple, le coprocesseur possède une portion de mémoire OCRAM dédiée, lui garantissant une autonomie complète vis à vis du reste du système. Le coprocesseur sur lequel se portera notre étude dans le chapitre 3 est intégré de manière similaire dans son SoC.

1.3.5 Les contrôleurs de débogage matériel

Le débogage matériel tel que le JTAG ou le SWD offre une multitude de fonctionnalités qui permettent aux concepteurs et aux développeurs de détecter au plus vite des erreurs dans les systèmes. Mais ces outils de débogage représentent également une menace pour la sécurité des données si leurs accès ne sont pas contrôlés. De nos jours, les fabricants ont commencé à prendre conscience de ces menaces [1, 119] et pour parer à cela, soit ils utilisent des solutions fournies par ARM®, soit ils implémentent leurs propres solutions, soit ils combinent les deux.

La solution ARM CoreSight® permet d'une part d'avoir un kit complet pour faire du débogage et, d'autre part, propose un moyen de contrôle des accès au débogage à travers l'interface *CoreSight authentication interface* [137]. Des signaux envoyés aux différents blocs IP autorisent ou verrouillent les fonctionnalités de débogage. Cependant, cette solution ne se limite qu'aux systèmes IP d'ARM. C'est pourquoi, afin de verrouiller le débogage de l'ensemble des parties d'un SoC, certains constructeurs intègrent des blocs matériels qui contrôlent les accès depuis l'extérieur. Ces contrôleurs servent de porte d'entrée aux différents protocoles de débogage et proposent souvent plusieurs modes de verrouillage (ex. accès -libre, -par authentification , -verrouillé [126]). La combinaison des IP ARM CoreSight® avec des contrôleurs d'accès permet un double contrôle de la sécurité des accès au débogage.

Le chapitre 5 est consacré aux systèmes de débogage. Dans celui-ci, nous reviendrons sur différents mécanismes de sécurité liés au débogage matériel. Ce sera également l'occasion de discuter de leurs faiblesses et des risques encourus.

1.3.6 La détection

Le but de la détection est de surveiller des variations inhabituelles de grandeurs physiques du système, qu'elles soient accidentelles ou volontaires. Le second cas arrive dans le cadre des attaques matérielles utilisant les chemins d'attaque décrits dans la section 1.2.4. En effet, pour arriver à leurs fins, les attaquants exploitent des grandeurs physiques liées aux systèmes afin de faire fonctionner ces derniers en dehors des plages de valeurs prévues par le cahier des charges. Ceci produit des comportements non prévus qui peuvent être maîtrisés pour réaliser des attaques. Toutefois pour prévenir cela, les fabricants utilisent des procédés de détection basés sur des capteurs [129]. De nombreuses grandeurs physiques peuvent être surveillées grâce aux différents types de capteurs (voir la liste des grandeurs physiques donnée par le tableau 1.1 de la Section 1.2.4) :

- Des capteurs de variation soudaine de tension. Ceci permet de limiter les attaques par *glitch* sur l'alimentation.

- Des capteurs de mesure de temps. La surveillance temporelle peut être faite à l'aide de *timer* et de *watchdog* qui mesurent et détectent des anomalies d'exécution de processus (voir Annexe A.6.3).

- Des capteurs de variation de fréquences d'horloges pour mesurer les tentatives de perturbation fréquentielles.

- Des capteurs de présence de la coque protectrice en métal pour signaler toute tentative de dégradation des protections physiques. Différents boîtiers protecteurs existent et sont rappelés dans l'annexe A.1.

- Des capteurs de températures qui ont la tâche d'informer le système que les plages de températures dans lesquelles il fonctionne sont celles définies par le cahier des charges.

Cette liste n'est pas exhaustive et des capteurs mesurant des variations pour chaque grandeur physique peuvent être imaginés. Cependant, en ce qui concerne les SoC, tous les risques ne sont forcément pas considérés et certains types de capteurs ne sont donc pas intégrés.

1.3.7 L'isolation matérielle des processus

La démocratisation d'appareils électroniques portatifs tels que les ordinateurs portables ou les *smartphones* a produit de nouveaux comportements. Des pratiques comme *Bring Your Own Device* (BYOD) ou apportez vos appareils personnels en français, se sont développées au seins des entreprises. Avec ces pratiques, les collaborateurs d'une entreprise on tendance à utiliser leur matériel professionnel à des fins personnelles et de manière réciproque, utiliser leur matériel informatique personnel à des fins professionnelles. Par conséquent, sur un même appareil, des données d'entreprises telles que les e-mails et divers documents sensibles évoluent dans le même contexte applicatif que des programmes tierces dont les sources ne sont pas forcément connues, comme pour certains jeux ou applications multimédias. Ceci présente un haut risque pour la sécurité des données de l'entreprise.

Afin d'éviter la corruption de ces données par un programme malveillant, les concepteurs de SoC ont développé des architectures visant à séparer différents processus s'exécutant sur un système. De cette manière, les données et processus propres à une entreprise sont isolés et protégés du reste des programmes. Ce type d'architecture se base sur le principe de la MMU et l'étend à tout le système en cloisonnant un jeu de processus donnés à tous les niveaux du système. Lorsque ce principe est appliqué à un programme, les ressources du système auxquelles il a accès sont limitées à une liste prédéfinie. Ceci évite que ce programme atteigne des données auxquelles il ne devrait pas.

La technologie ARM TrustZone® est aujourd'hui la plus répandue pour les solutions de partitionnement matériel des systèmes sur puce. Avec cette architecture, deux systèmes d'exploitation peuvent être exécutés en parallèle sur un même processeur de manière isolé l'un de l'autre. Un nouveau mode de privilège permet de changer de contexte d'un système d'exploitation à l'autre. Dans la littérature, ces contextes forment deux « mondes », l'un sécurisé et l'autre pas. Ces mondes sont séparés de manière matérielle et logicielle par différents IP qui contrôlent les accès aux ressources du système (mémoires, bus, périphériques, débogage, etc.). L'information du niveau de privilège du monde sollicitant la ressource est propagée à travers tout le système grâce à des signaux spécifiques [137]. Le monde sécurisé exécute les programmes dont les données doivent être protégées alors que le monde non sécurisé exécute le reste.

Les *Trusted Execution Environment* (TEE) sont des systèmes d'exploitations sécurisés qui cohabitent en parallèle d'autres OS non sécurisés, les *Rich* OS. Ils sont hermétiquement

isolés et doivent se partager les ressources du système de manière séquentielle. Le changement de contexte est géré par le processeur au travers d'interruptions et l'échange de données s'effectue avec des API dédiées. Une partie de ces TEE s'appuient sur l'architecture TrustZone® pour renforcer l'isolation des processus de manière matérielle (voir Fig. 1.12). Le chapitre 4 expose le fonctionnement de ces mécanismes et étudie leur résistance vis-à-vis des attaques matérielles.

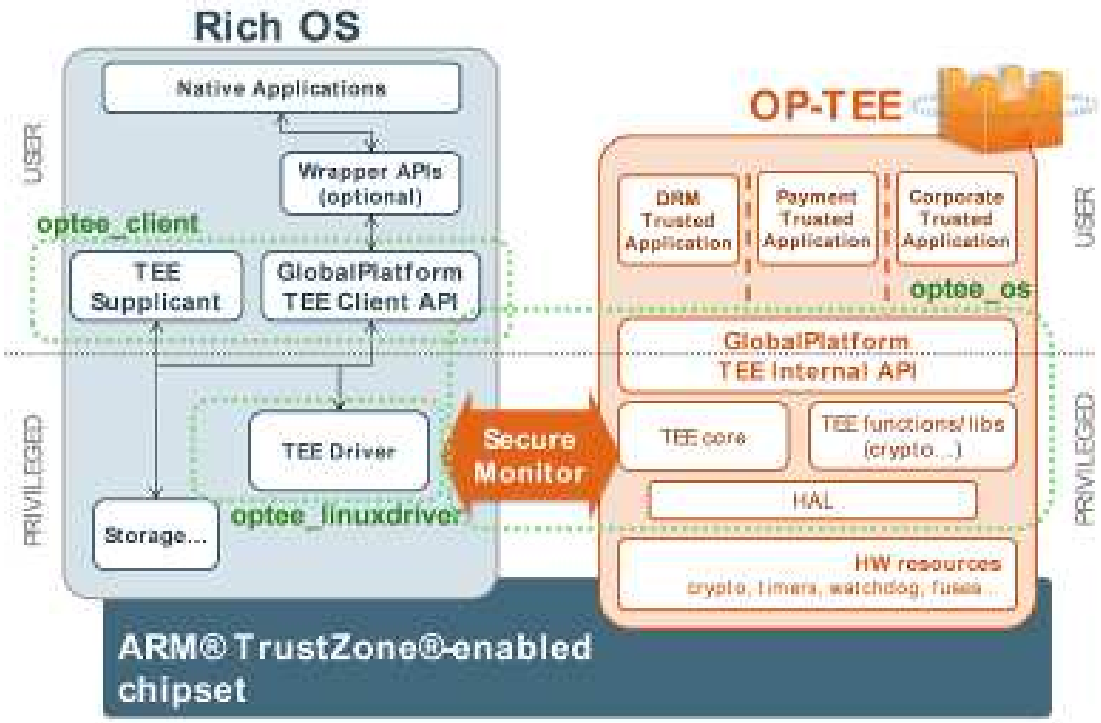


FIGURE 1.12 – Exemple d'architecture d'isolation matérielle de processus [11]

11. Source : https://www.linaro.org/blog/op-tee-open-source-security-mass-market/

1.4 Synthèse et positionnement des travaux de cette thèse

Dans ce chapitre nous avons décrit le contexte dans lequel s'inscrivent les travaux présentés dans ce manuscrit. Nous avons vu que dans le domaine de l'intégration des circuits intégrés, les SoC sont le niveau supérieur en terme de complexité à celui des microcontrôleurs. Leurs architectures modulaires basées sur l'agglomération de modules soumis à la propriété intellectuelle répond à une demande du marché notamment celle des téléphones mobiles et des objets connectés. Cependant, nous avons également vu que la diversité technologique de ces systèmes multiplie les chemins d'attaques et que pour ces raisons les concepteurs de SoC intègrent différentes solutions de sécurités.

Etant donné l'étendue du sujet, nous n'avons pas pu balayer l'ensemble des SoC et des modules de sécurité qu'ils intègrent face aux attaques existantes. Nous nous sommes donc imposé les contraintes suivantes :

- Nos études se sont concentrées sur des modules suffisamment génériques pour qu'ils puissent être les plus représentatifs possibles. Compte tenu du nombre de modèles de SoC présents sur le marché, nous en avons sélectionné quelques uns intégrant des technologies largement répandues pour servir de véhicules de tests.

- Seules sont étudiées les attaques non invasives qui ne demandent pas de lourdes modifications des circuits pour être appliquées (voir Définition 4). Cela permet d'étudier des attaques relativement simples, moyennement onéreuses, exécutables par une grande catégorie d'attaquants et par conséquent plus représentatives des risques.

Nous avons étudié la résistance des mécanismes de sécurités face aux attaques matérielles en considérant ces contraintes, les différents chemins d'attaques existants ainsi que les difficultés à les exploiter sur des SoC. L'objectif étant de répondre aux question suivantes :

1. Est-il possible de mettre en place des attaques matérielles sur les SoC ?
2. Peut-on proposer une méthodologie pour évaluer la vulnérabilité des SoC face aux attaques matérielles ?
3. Quelle est l'efficacité de telles attaques ?

Nous répondrons à ces différentes questions tout au long du manuscrit. Dans le prochain chapitre nous décrivons des différents outils que nous avons dû adapter pour mener à bien cette étude.

Chapitre 2

Mise en œuvre expérimentale

Dans ce chapitre nous décrivons la méthode et les outils que nous avons dû mettre en place afin de pouvoir étudier les attaques matérielles sur des processeurs complexes embarqués. Dans un premier temps, nous proposons une méthodologie générique que nous avons développée tout au long de notre étude pour évaluer la faisabilité d'attaque matérielles. Une réflexion est effectuée sur les grandeurs physiques à utiliser dans notre contexte et quels sont les paramètres impliqués. Ensuite nous décrivons comment nous avons adapté les bancs habituellement utilisés pour l'évaluation de cartes à puce afin d'optimiser nos manipulations.

Sommaire du chapitre

2.1 Méthodologie expérimentale

Notre étude cherche à caractériser les vulnérabilités des SoC face aux différents types d'attaques matérielles dans un contexte donné. Pour mener cette étude de manière rigoureuse, nous avons élaboré une méthodologie qui a deux objectifs principaux. Le premier est de définir un unique schéma d'expérimentation permettant de pouvoir comparer la sensibilité de différents systèmes face aux mêmes types d'attaques, idéal pour évaluer le niveaux de sécurité. Le second est d'avoir une méthode qui permette de sélectionner au fur et à mesure les différents paramètres nécessaires à la mise en place des différentes attaques.

Trois étapes principales composent cette méthodologie. Une première phase d'analyse suivie de deux étapes d'attaques *side-channel* et *fault injection*. L'attaque *side-channel* est appliquée lorsque le module ciblé manipule des données secrètes qu'un attaquant peut extraire par ce moyen. La figure 2.1 schématise le déroulement de cette procédure appliquée à un module M que dont on souhaite caractériser les vulnérabilités. Chaque étape est développée dans les paragraphes suivants.

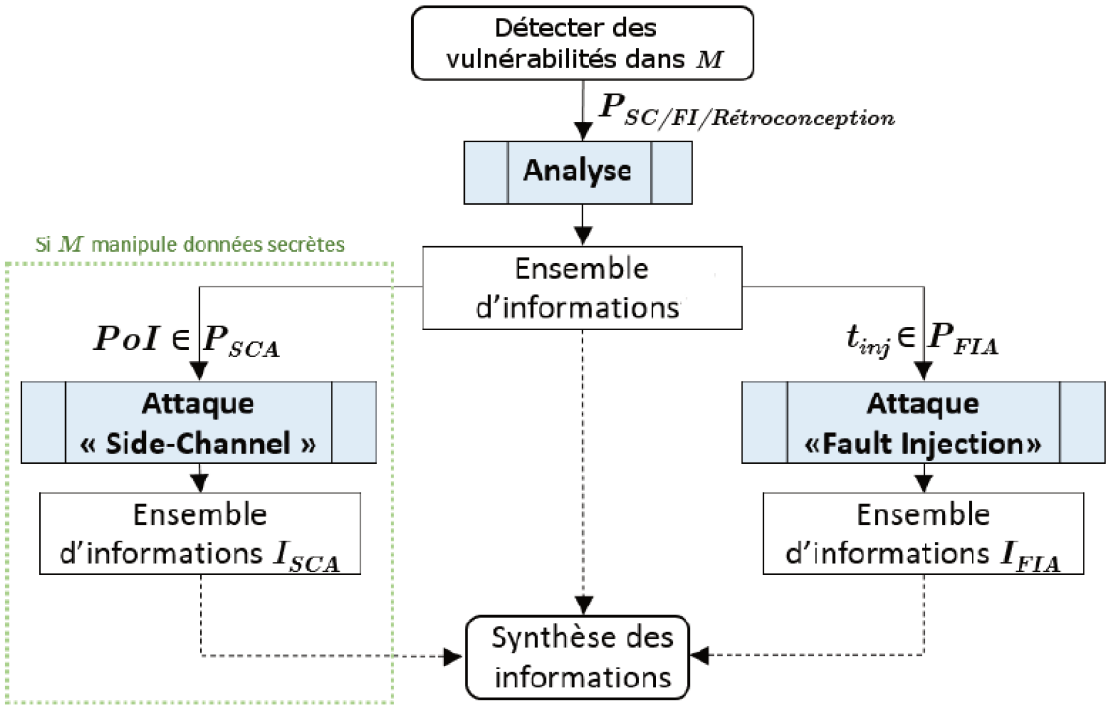


FIGURE 2.1 – Schéma de la procédure appliquée durant nos évaluations pour les différents modules M impliqués dans la sécurité

2.1.1 L'analyse

La première étape d'analyse permet d'obtenir des informations vis-à-vis du processus attaqué. Cette phase peut combiner l'application de techniques de rétroconception (voir section 1.2.2.1 du chapitre 1), d'analyses *side-channel* ou de perturbations. Il s'agit d'une étape essentielle qui fournit les informations nécessaires à la mise en place des autres étapes d'attaques matérielles. Que ce soit pour des attaques par canaux cachés ou par injection de faute, cette phase d'analyse permet de comprendre le déroulement des différentes tâches d'un processus ciblé et, le cas échéant, de savoir où attaquer. Par exemple, pour un module cryptographique, cette étape consiste à repérer la répétition de certains motifs caractéristiques sur les mesures physiques comme les rondes d'un AES ou l'exponentiation rapide d'un RSA afin de pouvoir localiser spatio-temporellement les activités de l'algorithme ciblé.

On note $P_{SC/FI/Rétroconception}$, l'ensemble des paramètres à définir pour mener à bien cette étape. A l'issue de celle-ci, un ensemble d'informations caractérise l'exécution du processus ciblé par le module M. Cet ensemble fournit une partie des informations nécessaires au paramétrage des deux prochaines étapes (ex. les instants à attaquer).

2.1.2 L'attaque side-channel

Cette étape ne s'applique que pour les modules manipulant avec des données secrètes et évalue la résistance de ceux-ci vis-à-vis des attaques *side-channel*.

Ces attaques visent principalement l'implémentation d'algorithmes cryptographiques (voir section 1.2.2.3 du chapitre 1). Pour pouvoir les appliquer, un attaquant doit connaître l'algorithme qui est en train de s'exécuter et maîtriser une partie du module attaqué afin de pouvoir le solliciter un minimum. L'identification de l'algorithme est réalisée à partir de l'ensemble d'informations issues de l'analyse précédente. La sollicitation du module est faite de manière à pouvoir manipuler des données et effectuer des mesures qui seront ensuite analysées afin d'établir une relation entre la variation de la grandeur mesurée et le secret impliqué dans la donnée traitée. A l'issue de diverses manipulations, l'attaquant obtient d'éventuelles informations sur la donnée secrète qui est le plus souvent une clef cryptographique.

Les instants durant lesquels un attaquant doit focaliser ses mesures sont une information également issue de l'étape d'analyse. Dans ce contexte, on appelle Point d'Intêret (PoI) l'ensemble des points temporels pour lesquels le secret attaqué est manipulé et des informations à son sujet sont potentiellement accessibles. On parle de « fuite d'informations » lorsque des données sont effectivement obtenues. La sélection des PoI est un paramètre indispensable parmi l'ensemble des paramètres P_{SCA} à définir pour procéder à une attaque *side-channel*.

A l'issue de cette étape, un ensemble d'informations I_{SCA} caractérise la fuite d'informations du module M vis-à-vis des attaques *side-channel*. En fonction des grandeurs physiques utilisées, cet ensemble souligne spatialement ou temporellement les zones et les paramètres pour lesquelles les informations sensibles manipulées par M sont potentiellement disponibles pour un attaquant *i.e.* cet ensemble met en valeur les vulnérabilités de M face aux attaques *side-channel*.

2.1.3 L'attaque fault injection

L'injection de faute peut s'utiliser pour deux catégories d'attaques : les attaques par analyses de fautes et les attaques par perturbations (voir section 1.2.2.2 du chapitre 1.)

Lorsque le module M évalué est impliqué dans des calculs cryptographiques, les deux catégories d'attaques peuvent s'appliquer et ont pour objectif d'aider à extraire le secret en réduisant la complexité algorithmique. La première catégorie de perturbations pour l'analyse de fautes est destinée à effectuer des attaques DFA (voir section 1.2.2.2.3 du chapitre 1). Le second type de perturbations vise à détourner le flot d'exécution du processus de chiffrement pour le rendre incomplet. Par exemple, perturber un processus de chiffrement AES de manière à récupérer le *state* à l'issue de tours intermédiaires, en se basant sur les messages en clair utilisés, diminue dangereusement le nombre d'hypothèses à effectuer sur la clef secrète pour retrouver sa valeur.

Lorsque le module M évalué est un mécanisme de verrouillage d'une fonctionnalité quelconque, les perturbations sont utilisées essayer de contourner cette sécurité. Ces mécanismes peuvent être, par exemple, des vérifications de codes PIN ou des contrôles d'identité. L'attaque consiste à perturber le test qui vérifie si un processus a le droit de se voir octroyer un privilège ou pas. Par ce biais, un attaquant force le test à autoriser une action en principe interdite. Il s'agit d'attaques par élévation de privilèges.

Dans un premier temps, les perturbations sont appliquées pour détecter la configuration des paramètres qui amèneront le système M à se comporter de manière anormale. C'est une phase de rétroconception qui souligne les valeurs d'une grandeur physique pour lesquelles une perturbation aura une répercussion sur le fonctionnement du module M (ex. intensité électrique, instant et durée de la perturbation). Dans un second temps, les perturbations servent à mener l'attaque à proprement parler.

Pour les attaques par faute, l'instant t_{inj} durant lequel la perturbation doit être générée est un paramètre essentiel qui a lui aussi été déterminé à partir des informations issues de la première étape d'analyse. Ce paramètre temporel t_{inj} fait partie de l'ensemble des paramètres P_{FIA} à définir pour procéder à une attaque par injection de faute.

A l'issue de cette étape, un ensemble d'informations I_{FIA} caractérise la sensibilité du module M vis-à-vis des attaques *fault injection*. En fonction des grandeurs physiques exploitées, cet ensemble souligne spatialement ou temporellement les zones et les paramètres pour lesquelles le module M est sensible face aux injections de faute.

2.1.4 L'étape de synthèse des informations

Cette étape permet d'avoir une vue d'ensemble du comportement du module testé vis-à-vis des différents types d'attaques matérielles. Elle permet également de s'assurer de la pertinence des données observées en recoupant les résultats des manipulations effectuées. C'est durant cette étape que sont comparées les informations de la première analyse, I_{SCA} et I_{FIA}, comme par exemple les PoI avec les mesures *side-channel* $\in I_{SCA}$ et l'apparition des différentes perturbations $\in I_{FIA}$.

Avec ces analyses, une conclusion peut être donnée quant à la vulnérabilité du module M face aux attaques matérielles, à la pertinence des données obtenues et à la nécessité d'expérimentations complémentaires.
L'application de cette méthodologie sera illustrée par les prochains chapitres de cette étude.

2.2 Réflexion sur les grandeurs physiques exploitables

Comme nous l'avons décrit tout au long du premier chapitre, les systèmes que nous cherchons à étudier ont des architectures hétérogènes complexes qui manipulent une multitude de grandeurs physiques exploitables pour des attaques matérielles. La simplicité et la faisabilité des attaques dépend de la grandeur exploitée. Afin de pouvoir appliquer notre méthode dans un temps raisonnable et d'étudier des attaques susceptibles d'être effectuées par une majorité d'attaquants, nous nous sommes imposés des contraintes comme, notamment, l'étude d'attaques non invasives uniquement (voir Section 1.4). Le Tableau 2.1 reprend les chemins d'attaque mis en valeur dans la section 1.2.4 du précédent chapitre en jugeant si celles-ci sont adaptées aux objectifs que nous nous sommes fixés.

Chemin d'attaque	**Attaque invasive**	**Attaque locale**	**Difficultés pour les SoC**	**Adapté**
Alimentation électrique	Non	Non	Gestion des alimentations. Capacités de découplages. (▶ 1.2.4.1)	Non
Probing	Oui	Oui	Architectures multi-couches complexes (▶ 1.2.4.1)	Non
Rayonnement électromagnétique	Non	Oui	Matériaux. Paramètres à définir. (▶ 1.2.4.2)	Oui
Temps d'exécution ou fréquence d'horloge	Non	Oui/Non	Horloges internes et contre-mesures. (▶ 1.2.4.3)	Non
Lumière	Oui	Oui	Boitiers et packaging. (▶ 1.2.4.4)	Non
Température	Oui/Non	Oui/Non	Précision, Cold Boot Attacks. (▶ 1.2.4.5)	Non

Tableau 2.1 – Evaluation des grandeurs physiques exploitables pour des attaques matérielles dans notre contexte. Les sections du chapitre 1 détaillant les difficultés dans le cas des SoC sont désignées par : (▶ x.x.x.x)

Ainsi, pour les systèmes sur puces, compte tenu des avantages et des contraintes inhérentes aux différents chemins d'attaques exploitables, la grandeur la plus appropriée pour effectuer une attaque de manière non invasive est le rayonnement électromagnétique. En effet, cette grandeur présente plusieurs avantages :

- Elle peut à la fois être utilisée pour des attaques SCA [114, 57, 9, 117] et des FIA [87, 100, 24].

- Elle dépend de l'espace et permet donc de restreindre la zone ciblée. Pour les SCA, les rayonnements EM en provenance d'un endroit précis du SoC peuvent être mesurés

et liés à un module IP en particulier. Pour les FIA, elle est utilisée afin de générer des perturbations locales, autrement dit elle permet de cibler la zone à perturber.

- Dans une majorité de cas, elle permet d'attaquer un SoC à travers son boîtier quelle que soit l'orientation de la face active de la puce.

C'est donc cette grandeur que nous exploiterons tout au long de cette étude pour effectuer nos attaques matérielles sur des SoC. En contrepartie, cela implique l'ajustement de plusieurs paramètres que nous détaillons dans les parties suivantes de ce chapitre.

2.3 Paramètres à définir pour nos manipulations

L'utilisation des champs électromagnétiques pour des attaques requiert l'ajustement de nombreux paramètres. Ceux-ci sont résumés dans le tableau 2.2 et notre approche pour les définir est donnée par la suite.

Paramètres	**Attaque EM...**	
	...par canal auxiliaire (§2.3.1)	**...par injection de faute** (§2.3.2)
Sonde	Forme/orientation. (§2.3.1.1)	Forme/orientation. (§2.3.2.1)
Localisation temporelle	Identifier l'instant du signal contenant l'information ciblée. (§2.3.1.2)	Identifier l'instant durant lequel la perturbation doit être injectée. (§2.3.2.2)
Localisation spatiale de la sonde	Qui maximise le RSB de l'information ciblée. (§2.3.1.3)	Qui perturbe le module exécutant le processus ciblé. (§2.3.2.3)
Temps et amplitude	Adaptés à la fréquence et l'ampleur des signaux. (§2.3.1.4)	Adaptés au type de faute désirée. (§2.3.2.4)

Tableau 2.2 – Synthèse des différents paramètres à définir pour mettre en place des attaques matérielles exploitant les champs EM. La description de notre approche pour chaque point est détaillé dans les sections notées §x.x.x.x

2.3.1 Paramètres à définir pour une SCA-EM

La principale difficulté pour mener une attaque side-channel est d'acquérir des signaux dans lesquels les informations vis à vis du secret ciblé ne sont pas trop occultées par des signaux parasites. Il s'agit d'un problème d'optimisation d'un rapport signal à bruit (RSB). Une partie de ce bruit est inhérente à la chaîne de mesure et l'autre partie provient des modules du SoC actifs pendant l'exécution du processus ciblé. En effet, ces modules émettent des rayonnements EM qui ne sont pas forcément en rapport avec le processus attaqué et par conséquent, brouillent les informations qui nous intéressent. C'est pourquoi, afin d'optimiser le RSB des mesures, il est nécessaire de déterminer de manière adéquate les différents paramètres suivants :

2.3.1.1 Le choix de la sonde de mesure

Les sondes de mesures EM sont des outils qui utilisent l'induction électromagnétique pour mesurer des variations de champ EM. Elles sont constituées d'un bobinage électriquement conducteur enroulé autour d'un noyau ferromagnétique. Lorsque ces sondes sont soumises à des variations de champs EM, on observe l'apparition d'un courant aux bornes de la bobine. Le choix de la sonde n'est pas trivial. Il n'y a pas aujourd'hui de méthode

formelle qui permette de sélectionner une sonde (forme, taille, orientation, bande passante, etc.) en fonction des blocs IP ciblés dans un SoC. Toutefois, certaines règles peuvent être appliquées.

- Une sonde ayant des dimensions excessives par rapport au module ciblé n'aura pas un bon rapport signal à bruit. En effet, elle captera des émissions EM sur une plus grande superficie, ce qui augmente la quantité de signaux non liés avec l'information ciblée.
- De faibles dimensions de la sonde impliquent une faible amplitude des signaux mesurés. En outre, plus la taille de la sonde est petite par rapport à celle du SoC ciblé, plus le nombre de points à observer spatialement pour trouver le signal recherché sera élevé.
- Ne pas risquer d'atténuer les signaux recherchés en utilisant une chaîne de mesure avec une bande passante inférieure à la fréquence de travail du système ciblé. La sonde est le premier maillon de cette chaîne.

Les différentes expérimentations que nous effectuons au cours de ces travaux illustreront la recherche de ce paramètre en appliquant ces principes.

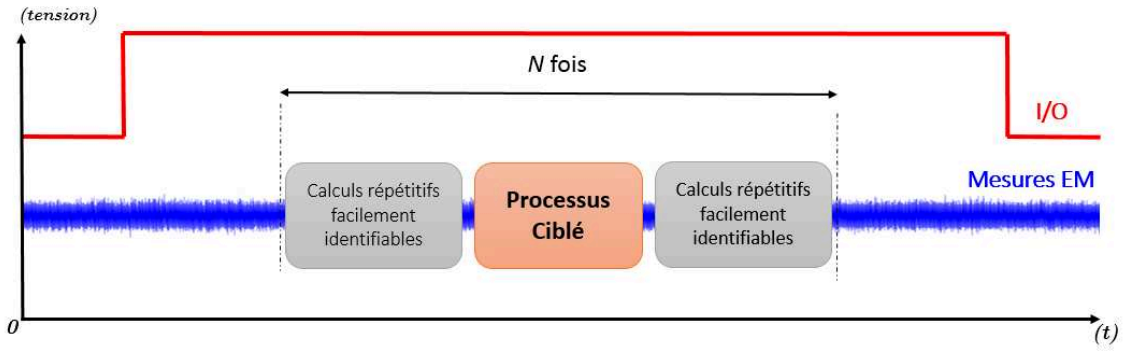


FIGURE 2.2 – Localisation temporelle de l'exécution d'un processus recherché.

2.3.1.2 La localisation temporelle

Les informations relatives au processus ciblé sont effectuées à des instants précis. Lorsque cela est possible, pour limiter la taille des signaux à observer, nous utilisons des repères temporels qui délimitent la période durant laquelle il est certain que le processus ciblé s'effectue. Pour mettre en place ces repères, des I/O disponibles sur le SoC et du code additionnel sont utilisés. L'objectif est de délimiter temporellement le signal recherché par des signaux simples à identifier. Par exemple, un port I/O tel qu'un GPIO émet un signal qui délimite la durée des opérations et des calculs répétitifs facilement identifiables sur des mesures EM, telles que des boucles incrémentales, sont introduits avant et après les opérations ciblées afin de les marquer. La répétition N fois du processus permet d'accentuer la signature du signal caractéristique ainsi créé. La figure 2.2 illustre cette méthode de localisation temporelle.

2.3.1.3 La localisation spatiale

Les champs électromagnétiques ont des composantes spatiales, donc leur variation en fonction d'une donnée manipulée sera dépendante de la position du bloc IP qui utilise cette dernière. Par conséquent, les informations obtenues avec une sonde EM sont dépendantes de la position de celle-ci vis-à-vis du SoC. D'autre part, étant donné que l'intensité du champ décroît à l'inverse du carré de la distance, on cherchera toujours à placer la sonde au plus près du silicium, donc au contact du boîtier plastique du SoC pour effectuer nos mesures. On parle d'analyse *side-channel* en « champs proche ». Ainsi, la localisation spatiale conjointement avec la sélection de la sonde s'effectue de la manière suivante dans nos expérimentations :

1. Nous effectuons un premier balayage manuel afin d'évaluer si le signal recherché est rapidement observable. Ceci sous-entend que la forme du signal doit être connue. L'utilisation des marqueurs temporels du paragraphe précédent est prévue à cet effet.

2. Si l'étape 1 est infructueuse, un balayage plus minutieux avec une sonde adaptée doit être envisagé. Ceci peut être réalisé de manière automatisée.

3. Enfin, un traitement statistique simple sur plusieurs traces (du type moyenne) permet de diminuer le bruit de mesure et de mettre en valeur l'information recherchée. Une synchronisation des traces sera toutefois nécessaire au préalable. Nous présenterons une partie de ces traitements dans les différents chapitres de ce manuscrit.

4. Si aucune information sur le processus ciblé n'est détectée, il faut recommencer les manipulations avec une autre sonde et de nouveaux paramètres de mesure ou envisager la non-faisabilité de l'attaque avec les moyens mis en place.

Les différents modules attaqués durant notre étude permettent d'illustrer l'ensemble des points cités dans ce paragraphe.

2.3.1.4 Fréquence et amplitude d'échantillonnage

Le théorème de Shannon simplifié énonce que, lors de la numérisation d'un signal, le nombre d'échantillons par unité de temps doit être supérieur au double de la fréquence maximale de ce signal. De plus, le choix des différents éléments de la chaîne de mesure doit prendre en compte leur bande passante respective. Les éléments doivent tous avoir une bande passante supérieure à la fréquence de fonctionnement du système observé.

D'autre part, il peut exister d'importants écarts d'amplitude entre des signaux mesurés à différents emplacements spatiaux, la difficulté étant d'adapter la sensibilité du système d'acquisition pour avoir une dynamique suffisante tout en conservant le moyen de comparer les différents points de mesure. Bien qu'un système auto-adaptatif soit la solution la plus efficace, sa mise en place peut s'avérer très chronophage.

2.3.2 Paramètres à définir pour une FIA-EM

Dans les attaques par injection de faute, la difficulté est de perturber un processus en particulier à un instant précis. L'utilisation des champs électromagnétiques permet de créer des perturbations locales. C'est un avantage pour notre étude si on considère la quantité de blocs intégrés dans les SoC. En contrepartie, les champs EM demandent l'ajustement de nombreux paramètres pour générer une perturbation efficace. Ceux-ci présentent une certaine analogie avec ceux à définir pour les SCA-EM :

2.3.2.1 Le choix de l'injecteur

Les sondes d'injection EM ou injecteurs EM sont des outils qui utilisent l'induction électromagnétique pour générer de brusques variations de champ EM. De nombreux paramètres interviennent dans la conception de ces injecteurs afin de maximiser la génération de champs EM [97]. De manière similaire aux sondes de mesures, elles sont constituées d'un bobinage électriquement conducteur enroulé autour d'un noyau ferromagnétique. Lorsque qu'un courant électrique circule dans la bobine, on observe l'apparition d'un champ EM à proximité de celle-ci. La difficulté qui se présente ici est de sélectionner la sonde qui génère un champ à même de perturber le module ciblé. Sachant que les différents courant induits par les lignes de champ sont perpendiculaires à celles-ci, la forme de l'injecteur est prépondérante. De même que pour le choix de la sonde d'observation, il n'y a pas de méthode formelle qui permette de sélectionner un injecteur particulier. Cependant, des règles similaires peuvent être appliquée :

- Un injecteur ayant des dimensions excessives par rapport au module ciblé dans un SoC génèrera un champ dispersé qui risque de perturber l'ensemble du processeur.
- A l'inverse, un injecteur ayant des dimensions insuffisantes pour créer un champ à même de traverser les différentes couches (boîtier/métal) ne parviendra pas à générer un courant perturbateur dans les parties métalliques du circuit. De plus, les faibles dimensions de celui-ci augmenteront le nombre de points spatiaux à tester pour trouver une zone sensible.
- Procéder par de premières injections d'impulsions EM en balayant manuellement la surface du SoC avec une puissance suffisante pour produire des « plantages » complet de la cible. Ceci donne des informations sur l'efficacité de la sonde choisie et sur l'emplacement des zones sensibles. Aviser sur l'intensité afin de définir si des perturbations exploitables peuvent être produites dans les zones localisées.
- Effectuer des premières injections EM avec des sondes présentant un maximum de symétrie afin de limiter les paramètres d'orientation.

2.3.2.2 La localisation temporelle

L'efficacité d'une perturbation dépend du moment durant lequel elle est générée. C'est pourquoi il est impératif de connaître l'instant adéquat à la génération d'une perturbation EM. Cependant, de nouvelles contraintes s'appliquent ici. En effet, pour synchroniser la génération de l'impulsion EM avec le déroulement du processus, il n'est techniquement pas

possible de placer la sonde d'observation au même endroit que l'injecteur. L'observation en simultané avec la perturbation est compromise, mais deux alternatives sont envisageables :

1. La première, la plus précise, est de réussir à trouver un compromis entre une position de la sonde d'observation qui permette l'émission d'un signal de qualité et une position de l'injecteur qui produise des perturbations efficaces. En utilisant le repérage temporel de l'observation *side-channel* section 2.3.1.2, il est possible d'identifier dynamiquement l'instant d'injection de faute. Même si l'instant varie légèrement, l'ajustement de l'injection peut se faire en temps réel. La précision temporelle de la faute est maximum et le délai de réaction du dispositif d'injection devient alors un paramètre primordial.

2. Lorsque la première solution n'est pas applicable, il est possible de procéder en deux temps. Une première phase de mesure *side-channel* consiste à estimer l'instant de l'injection. En se basant sur le signal I/O émis par le SoC avant chaque exécution du processus ciblé, il est possible d'estimer la moyenne $\overline{t_{inj}}$ et l'écart type $\sigma_{t_{inj}}$ des temps d'injection des perturbations. La deuxième phase se déroule pratiquement en aveugle sur le banc d'injection EM. L'injection de la faute est programmée avec un délai après chaque signal I/O pour qu'elle ait lieu à $\overline{t_{inj}}$. Afin d'augmenter la probabilité d'injecter la perturbation durant l'instant ciblé et d'avoir un taux de succès suffisant, un balayage temporel peut être fait et le nombre de répétitions du processus d'injection peut être adapté en fonction de l'écart type observé.

2.3.2.3 La localisation spatiale

De la même manière que pour l'observation *side-channel*, l'emplacement spatial de la sonde qui injectera la pulsation électromagnétique est déterminant. Nous ne savons pas à l'avance quelles sont les parties susceptibles d'être à la fois réceptrices aux champs EM et perturbatrices pour un processus ciblé. C'est pourquoi il nous faut rechercher la ou les zones adéquates. Dans nos expérimentations, la recherche de la localisation spatiale conjointement à celle de la sélection de la sonde d'injection s'effectue de la manière suivante :

1. Régler l'intensité de l'impulsion EM de manière à générer des redémarrages du système ciblé. On évalue de cette manière la sensibilité aux perturbations EM du système attaqué. Puis, effectuer un premier balayage sur la surface du SoC afin de localiser les zones les plus sensibles. Enfin, expérimenter des valeurs intermédiaires de l'intensité EM sur les différentes zones identifiées. Compte tenu de l'évanescence des champs EM et de l'épaisseur des boîtiers plastique des SoC, on placera les injecteurs au contact de ces derniers.

2. Si l'étape 1 est infructueuse, un balayage plus minutieux avec un injecteur adapté doit être envisagé. Ceci peut être fait de manière automatisée.

3. Enfin, tester différentes valeurs, polarité de champs avec différents modèles d'injecteurs jusqu'à ce que l'effet escompté soit produit.

Cette approche est expérimentale et ne garantie pas le succès d'une attaque par injection de faute. Cependant, cela permet d'évaluer rapidement la complexité et la faisabilité d'une FIA sur un système donné.

2.3.2.4 Durée et amplitude de l'impulsion

Une perturbation consiste à faire fonctionner un système en dehors de ses conditions normales d'utilisation. En l'occurrence, cela consiste à lui injecter un surplus d'énergie au moment opportun. Ce surplus d'énergie est apporté par la génération d'impulsions EM au-dessus du circuit ciblé. Les impulsions sont générées par la combinaison d'un générateur d'impulsions de tensions électriques avec des injecteurs EM (décrits section 2.5 suivante).

Les deux paramètres configurables sur ce générateur sont la durée ΔT et l'amplitude U d'une impulsion de tension (voir figure 2.3). En particulier ce sont les temps de montée t_m et de descente t_d des impulsions qui définissent les variations de courant $\frac{\partial}{\partial t} i(t)$ dans le solénoïde de la sonde d'injection qui créent les impulsions EM. Par induction, des tensions « parasites » $\varepsilon_{cible} = -\frac{d}{dt}\Phi$ seront générées dans le circuit cible.

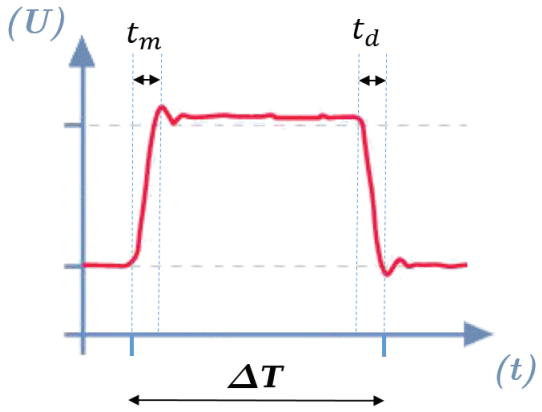


FIGURE 2.3 – Schéma d'une impulsion de tension créée par le générateur.

En ajustant les grandeurs U et ΔT, nous recherchons des valeurs pour lesquelles l'impulsion EM créée est suffisante pour entraîner une faute dans le processus ciblé. Cependant, générer une impulsion trop puissante risque de perturber excessivement le système jusqu'à un état instable qui aboutira à son redémarrage. Nous ajustons les valeurs ΔT et U pour nous placer juste à la limite au-delà de laquelle le système dysfonctionne complètement et redémarre. La limite qui à été expérimentalement établie est de 10% de redémarrages sur l'ensemble des injections EM.

Pour définir l'ensemble de ces paramètres, nous avons utilisé deux bancs automatisés dans nos expérimentations. Un banc de mesures électromagnétiques et un banc d'injections de perturbations électromagnétiques, tous deux décrits dans les sections 2.4 et 2.5 qui suivent.

2.4 Banc de mesure EM

Pour observer l'activité d'un SoC, nous avons opté pour une plateforme automatisée qui mesure les rayonnements électromagnétiques émis par celui-ci.

Le principe d'une telle plateforme est de mesurer les variations d'un champ EM liées à l'activité d'un circuit ciblé. Les champs EM mesurés sont principalement induits par la variation de la consommation de puissance électrique durant le changement d'état des portes logiques du circuit [102]. Les mesures sont effectuées à l'aide de sondes amplifiées en tension qui, par induction, transforment les champs EM en tensions mesurables par l'oscilloscope. L'analyse de ces dernières permet d'établir un lien avec un processus effectué par le circuit.

La plateforme utilisée pour nos travaux de recherche est représentée sur les figures 2.4 et 2.5. Il s'agit d'un banc qui a été conçu par l'équipe des Hardware Labs de Gemalto et qui est utilisé pour effectuer des évaluations de sécurité matérielle de cartes à puce. Afin de pouvoir utiliser ce matériel sur des SoC, il a fallut l'adapter. Par exemple, nous avons dû fabriquer un système de fixation pour les cartes de développement intégrant les SoC et avons remplacé le lecteur de carte à puce par un Raspberry Pi [12]. Dans l'ensemble, le banc que nous avons utilisé peut se résumer aux 6 parties suivantes (même numérotation que sur les Fig. 2.4 et 2.5) :

(1) **Les sondes de mesure EM.**

Pour cette étude, nous disposons de microsondes EM à champ proche de marque LANGER [13]. Elles sont composées d'un noyau torique en ferrite de quelques micromètres ($\varnothing$: de 100 à 500 μm) et d'un amplificateur intégré. Deux orientations du noyau sont disponibles, "HH" pour laquelle l'axe de symétrie du tore est normal à la surface à mesurer et "HV" pour laquelle cet axe est parallèle avec la surface à mesurer. Le second modèle présente un paramètre d'orientation spatial supplémentaire : l'angle $\varphi \in \left[0, \frac{\pi}{2}\right]$ (voir Fig. 2.6). Les bandes passantes des sondes sont comprises entre 1 et 6 GHz. Par effet d'induction, les champs EM créent des micro-courants (quelques microampères) qui, grâce à l'amplificateur intégré, sont transformés en tensions mesurables par l'oscilloscope.

(2) **L'oscilloscope numérique.**

L'oscilloscope numérique 8 bits acquiert et enregistre les signaux sur une mémoire externe afin de pouvoir les analyser par la suite. Quatre canaux sont disponibles pour acquérir des signaux simultanément ce qui permet de synchroniser la mesure de diverses informations comme par exemple : les signaux retournés par la sonde EM, l'alimentation du circuit ciblé, les signaux de « top » et *reset* qui permettent de déclencher l'acquisition des mesures. Avec une bande passante de 4GHz et une fréquence d'échantillonnage pouvant aller jusqu'à 40GS/s, l'oscilloscope est adapté aux fréquences de travail des SoC.

12. https://www.raspberrypi.org/
13. https://www.langer-emv.de/en/category/near-field-microprobes-h-and-e-field/69

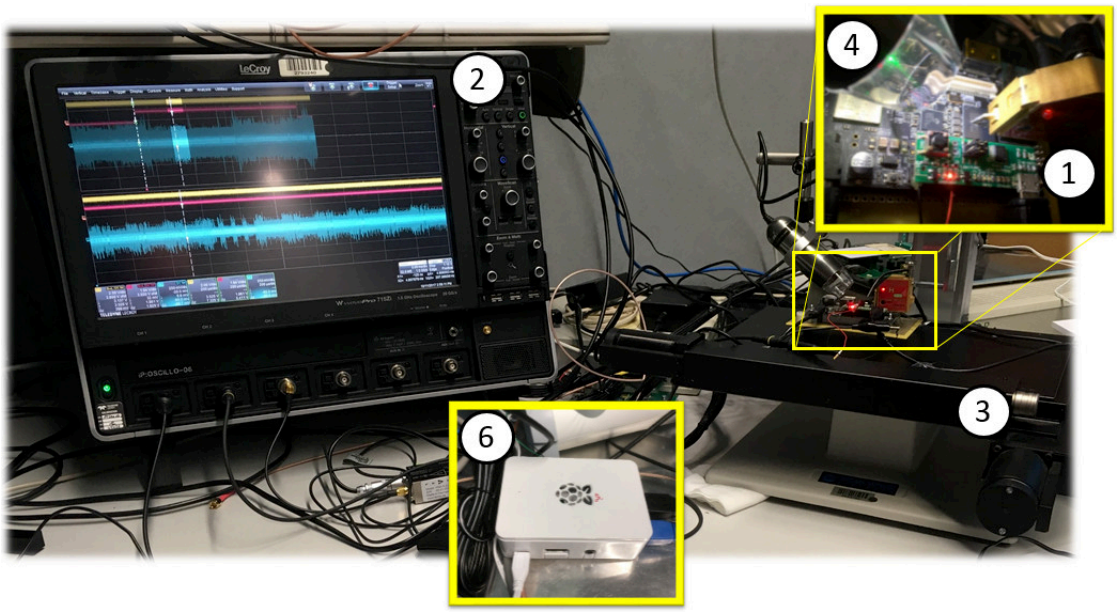


FIGURE 2.4 – Banc de mesure EM utilisé dans cette étude.

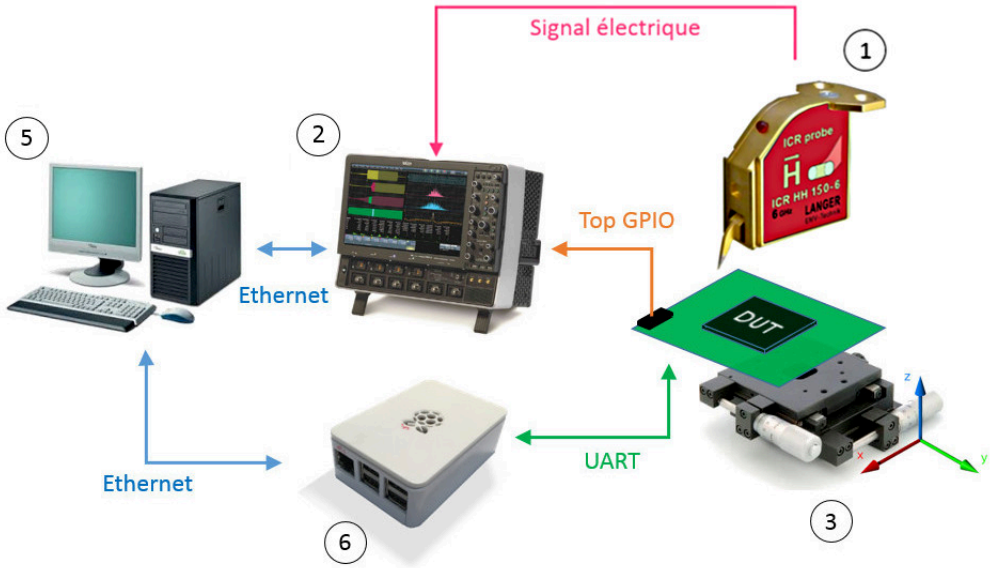


FIGURE 2.5 – Schéma simplifié du banc de mesure EM

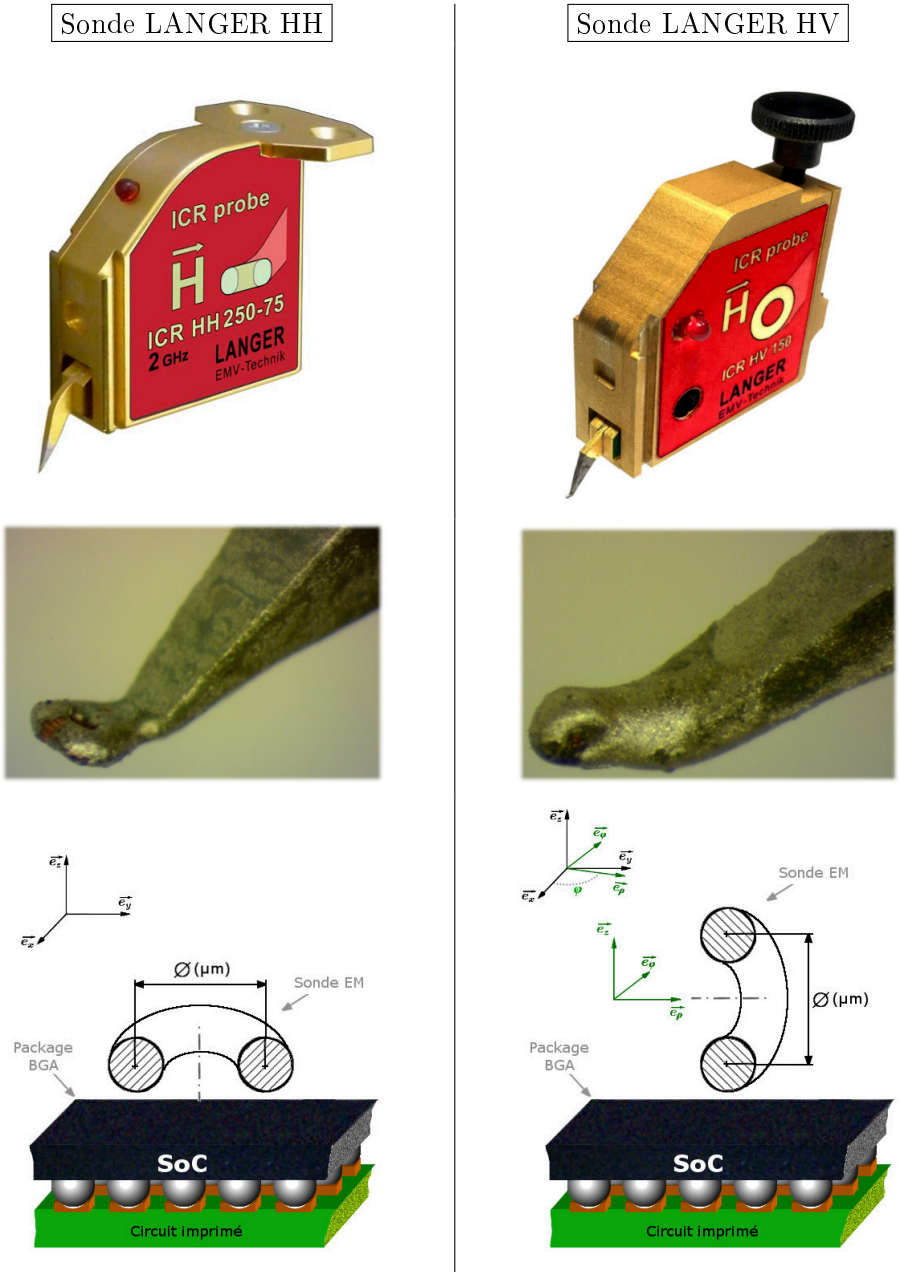


FIGURE 2.6 – Types de sondes de mesure EM à notre disposition

③ **Système de positionnement XYZ.**

Afin de se positionner spatialement à l'endroit du SoC que nous souhaitons observer, un système de 3 moteurs pas-à-pas permet de déplacer la sonde EM au micromètre près, selon les 3 axes de l'espace. Ces moteurs peuvent être directement commandés par des manettes ou bien dirigés par un ordinateur au travers d'un périphérique USB, des pilotes et un logiciel.

④ **Les caméras.**

Un système de caméras permet d'ajuster le positionnement de la sonde EM.

⑤ **Le PC de contrôle.**

L'ensemble du banc est relié par un réseau Ethernet TCP/IP au PC de contrôle. Un logiciel propriétaire développé par Gemalto permet d'automatiser les campagnes de tests. Les séquences d'évaluations sont définies par des scripts qui établissent l'ordre suivant lequel les instructions et les commandes sont échangées avec le système évalué. Ainsi, le logiciel permet, entre autres, d'envoyer des commandes et d'échanger des données. Les commandes peuvent être un redémarrage ou des instructions propres au système. Quant aux données, elles peuvent être, par exemple, les valeurs nécessaires à un calcul cryptographique. Les messages et les clés sont envoyés au circuit testé et celui-ci retourne le chiffré au PC. Ce logiciel permet également d'armer l'oscilloscope lors d'une instruction précise et d'effectuer des acquisition de courbes, d'effectuer des cartographies spatiales ainsi que des balayages temporels pour la génération de signaux de « top », etc. L'expertise de Gemalto dans la sécurité numérique a permis d'élaborer un logiciel optimisé pour procéder à l'évaluation de la sécurité des cartes à puce.

⑥ **Le module d'interfaçage avec le DUT.**

A l'origine, ce banc est destiné à évaluer des cartes à puce. Cependant, il possède pratiquement toutes les caractéristiques dont nous avons besoin. C'est pourquoi, au lieu de tout développer pour notre étude, nous avons préféré adapter le banc EM. Pour les cartes à puce, l'appareil qui fait l'interface entre le PC de contrôle ⑤ et la carte évaluée est un lecteur dénommé en interne « Lecteur CLIO ». Ce dernier alimente la carte, lui fournit un signal d'horloge le cas échéant et permet d'échanger des données avec elle, le tout suivant les normes dédiées aux cartes à puce : ISO 7816-XX [14]. Ce lecteur communique avec le reste du banc au travers d'un réseau Ethernet. Pour le remplacer, nous avons choisi d'utiliser un Rasperry Pi. Le Raspberry est un mini-ordinateur qui embarque un système d'exploitation Debian GNU/Linux. Cela nous permet d'une part, d'interpréter les commandes ISO 7816 pour simuler le comportement d'un lecteur CLIO du point de vue du PC de contrôle et, d'autre part , de communiquer avec le SoC à évaluer à travers le port série. De plus, les GPIO du Raspberry sont utilisés pour envoyer des signaux de `<Reset>` au SoC lorsque cela est demandé dans le script s'exécutant sur le PC.

14. https://www.iso.org/

2.5 Banc d'injection EM

Pour injecter des perturbations dans un SoC, nous avons opté pour une plateforme automatisée qui génère des impulsions EM à travers celui-ci.

Le principe d'une telle plateforme est de générer de brusques et intenses variations du champ EM à proximité du circuit ciblé. Ces variations sont ensuite transformées en courants par effet d'induction sur toutes les parties métalliques réceptrices du circuit, principalement les bus et les vias. Les courants ainsi créés perturbent les différents signaux et causent d'éventuelles fautes durant le fonctionnement du système.

La plateforme d'injection EM utilisée pour nos travaux de recherche est représentée sur les figures 2.7 et 2.8. Il s'agit d'un banc similaire à celui utilisé par le LIRMM (Laboratoire d'informatique, de robotique et de microélectronique de Montpellier) dans [98]. Ce banc est composé des différents éléments suivants :

(1) **Le générateur d'impulsions de tensions électriques.**

Le générateur d'impulsions tensions produit des impulsions de courant qui servent à générer les impulsions EM. Ces impulsions peuvent être paramétrées en amplitude (comprise entre ±400V avec un courant maximal de 16A) et en durée (de 6ns à 100ns). Pour chaque impulsion, le temps de montée est $\leq$ 2ns, pour un temps de descente $\leq$ 5ns. Un temps incompressible de $350ns$ de délai est présent entre le déclenchement de l'impulsion, et la génération de celle-ci. Ce délai est à prendre en compte dans les diverses manipulations.

(2) **Les sondes d'injections EM.**

Il s'agit de l'élément qui transforme les impulsions électriques en impulsions EM par effet d'induction. Nous disposons de plusieurs modèles de sondes d'injections avec ce banc. Elles ont été fabriquées à la main et chacune d'entre elle est composée d'un cœur de ferrite autour duquel est bobiné un petit fil de cuivre (voir Fig. 2.9). Lorsque les impulsions électriques traversent le bobinage de cuivre, par effet d'induction, des impulsions EM sont générées. Chaque sonde possède une forme de ferrite et un nombre de spires différents ce qui influence le type de champ créé [97]. Comme représenté sur la figure 2.9, les sondes cylindriques et coniques présentent une symétrie cylindrique par rapport à l'axe normal à la surface du SoC, ce qui implique que le champ produit affiche lui aussi cette symétrie. La précision spatiale est amoindrie mais la zone impactée est plus étendue. Les sondes dites « Oméga », sont pourvues d'une forme qui permet de concentrer le champ EM entre les deux extrémités. Cette forme permet d'avoir un champ orientable suivant l'axe normal à la surface du SoC.

Dans ce document, les injecteurs EM sont référencés de la façon suivante :

injecteur EM <type>-$\varnothing D.d$-Ns

Avec :

<type>	: Cyl. (Cylindrique), Con. (Conique), Ω (Oméga).
D	: le diamètre extérieur.
d	: le diamètre intérieur (uniquement pour les injecteurs Ω).
s	: le nombre de spires.

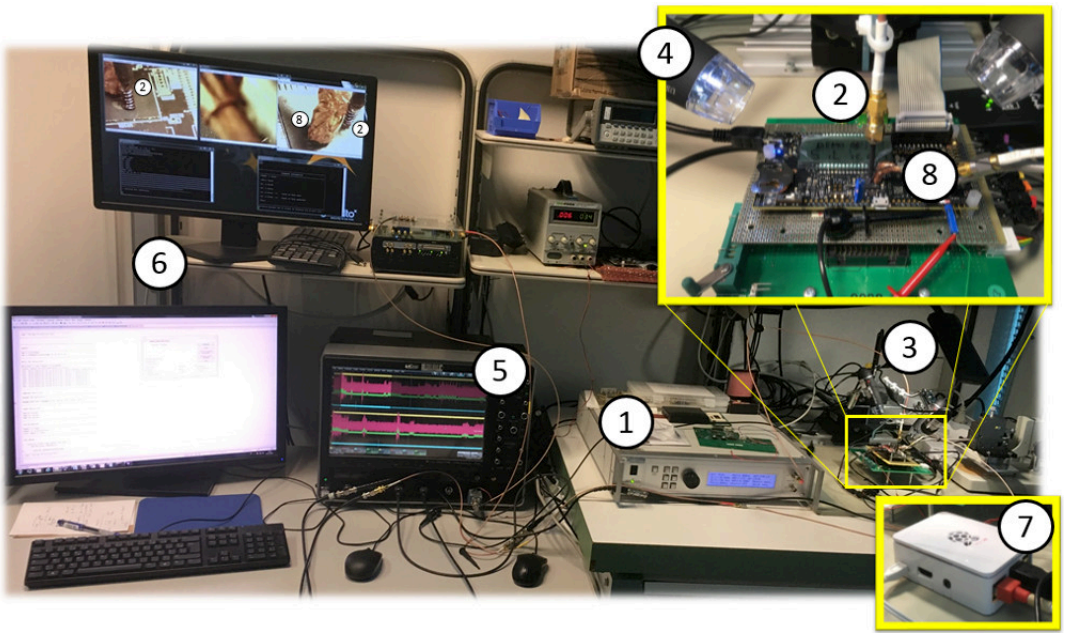


FIGURE 2.7 – Banc d'injection EM utilisé dans cette étude

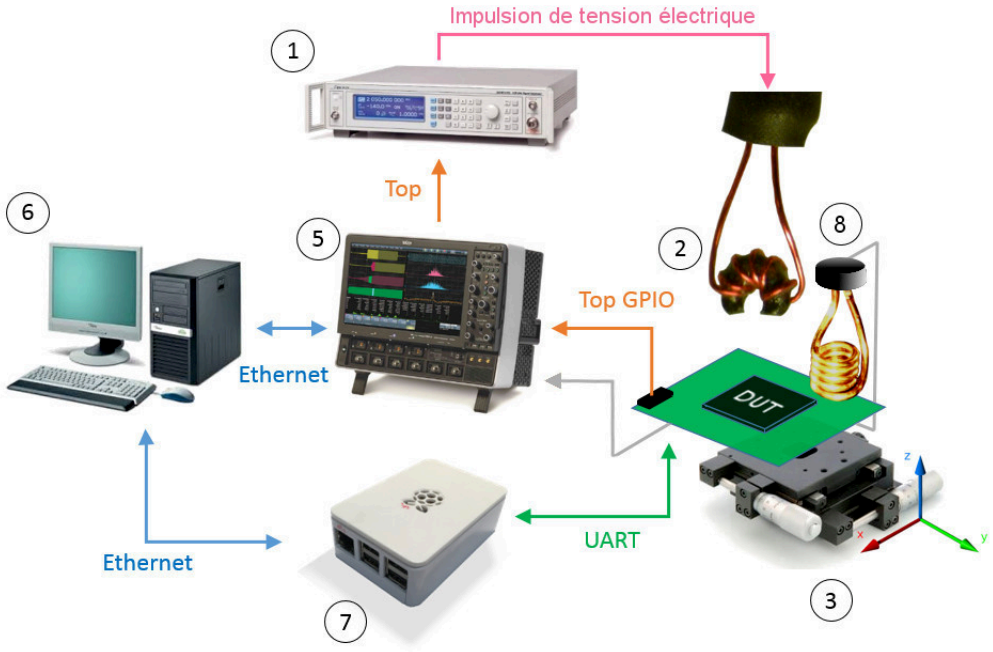


FIGURE 2.8 – Schéma simplifié du banc d'injection EM

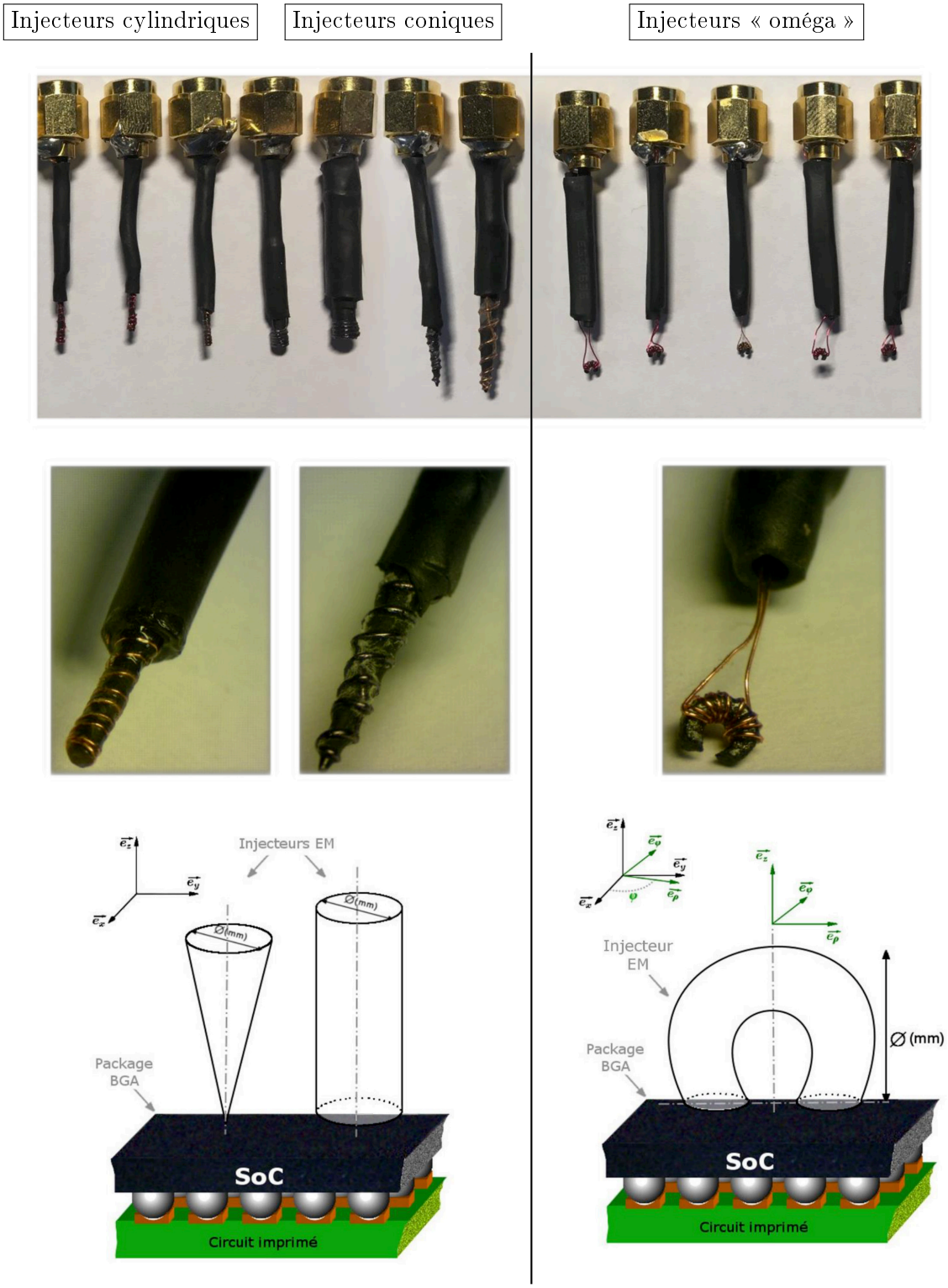


FIGURE 2.9 – Sondes d'injection EM à notre disposition

(3) **Système de positionnement XYZ.**

Comme pour le banc de mesure EM, un système de 3 moteurs pas-à-pas permet de positionner l'injecteur EM au micromètre près selon les 3 axes de l'espace.

(4) **Les caméras.**

Ici aussi des caméras sont utilisées pour ajuster le positionnement de l'injecteur.

(5) **L'oscilloscope numérique.**

Sur ce banc, l'oscilloscope numérique a deux fonctions. La première est de synchroniser le déclenchement de l'impulsion avec le fonctionnement du circuit. Lorsque l'oscilloscope reçoit un signal du circuit testé (un top GPIO par exemple), il génère un signal sur une de ses sorties qui servira de déclencheur au générateur d'impulsions (1). La deuxième fonction de l'oscilloscope est d'observer le comportement du circuit soumis aux impulsions. Pour cela, on place en parallèle de l'injecteur une sonde de mesure EM apte à résister aux impulsions infligées au circuit (8). Cette dernière acquiert les signaux reflétant l'activité du circuit perturbé.

(6) **Le PC de contrôle.**

Pour des raisons de praticité, nous avons utilisé le même logiciel que celui du banc de mesure EM. Ainsi, ce logiciel dédié s'exécute sur le PC qui pilote l'ensemble du banc. Le PC est relié à l'oscilloscope et au Raspberry par un réseau Ethernet TCP/IP.

(7) **Le Raspberry d'interfaçage.** Comme pour le banc de mesure EM, afin de conserver toutes les fonctionnalités développées pour l'évaluation des cartes à puce, nous avons choisit d'utiliser un Raspberry Pi (voir la description (6) de la section 2.4 consacrée au banc de mesure EM).

(8) **Sonde de mesure EM.** Cette sonde permet d'observer l'effet de l'impulsion EM. C'est une sonde capable de supporter la mesure des impulsions électromagnétiques intenses. Bien qu'elle ne possède ni une large bande passante, ni une bonne amplification des signaux, elle s'avère utile dans certains cas afin de pouvoir identifier le processus qui est en train de s'exécuter et ainsi ajuster l'instant de déclenchement des impulsions. L'information fournie par cette sonde couplée avec le GPIO permet d'augmenter la précision temporelle de l'injection de fautes.

2.6 Conclusion

Dans ce chapitre, nous avons présenté les outils que nous avons adaptés aux systèmes sur puce afin de pouvoir appliquer des attaques matérielles. Nous avons également décrit une méthodologie que nous proposons d'appliquer sur les modules de sécurité matérielle étudiés dans cet ouvrage. L'avantage de cette méthode est de proposer un schéma de test générique applicable à différents modules matériels.

Les SoC offrent de nombreuses grandeurs physiques exploitables pour des attaques matériel. Cependant, dans la section 2.2, nous expliquons pourquoi il est nécessaire de limiter nos expérimentations à l'utilisation des champs électromagnétiques comme vecteur d'attaque. Ainsi, avec l'exploitation de cette unique grandeur physique, nous pouvons procéder à des attaques *side-channel* et injections de faute non invasives. Ceci limite le nombre de manipulations à effectuer mais laisse de nombreux paramètres à ajuster. Leur liste est donnée section 2.3. C'est là que la méthode proposée section 2.1 de ce chapitre présente un second avantage. En effet, l'ordonnancement précis des différentes étapes de tests et l'acquisition d'informations au sujet des expérimentations permet d'optimiser l'ajustement des différents paramètres. Un guide permettant d'aborder la sélection des paramètres EM est proposé par les sections 2.3.1 et 2.3.2.

Enfin, pour détailler l'aspect pratique des différentes expérimentations, nous décrivons les bancs automatisés que nous avons utilisés tout au long de cette étude. Ceci illustre la complexité de l'application d'attaques matérielles utilisant les champs électromagnétiques sur des systèmes sur puce.

L'application de la méthodologie et l'utilisation des outils proposés dans ce chapitre seront illustrés dans les chapitres suivants.

Chapitre 3

Étude de la faisabilité d'attaquer l'exécution de traitements cryptographiques dans les SoC

Les attaques par canaux cachés et par injection de faute décrites dans le premier chapitre ont largement été appliquées sur des systèmes tels que les cartes à puce et les microcontrôleurs. En revanche, leurs applications sur des processeurs plus polyvalents tels que les systèmes sur puce ne sont pas encore très répandues dans la littérature scientifique. Les architectures et les nouveaux paramètres à considérer compliquent leur mise en place. Aujourd'hui, les études publiées portent uniquement sur l'aspect *side-channel*. Parmi celles-ci on peut citer les travaux de Genkin *et al.* [58] qui utilisent les champs électromagnétiques pour extraire des clefs de chiffrement utilisées par un protocole cryptographique s'exécutant dans le processeur d'un *smartphone*. On peut également citer l'impressionnant travail de Longo *et al.* [85] qui ont appliqué diverses techniques statistiques sur les émissions électromagnétiques pour extraire des clés de différentes implémentations cryptographiques.

L'objectif de ce chapitre est d'étudier la mise en place de telles attaques, mais également d'évaluer la faisabilité de perturber l'exécution de calculs cryptographiques dans un SoC, qu'ils soient logiciels ou matériels. Les principales unités traitant la cryptographie dans ces systèmes sont les processeurs principaux (CPU) et les accélérateurs cryptographiques matériels (C-HWA). Pour étudier des attaques sur ces modules, nous avons choisi de consacrer une première partie aux attaques d'implémentations cryptographiques logicielles et une seconde aux attaques d'implémentations matérielles. L' « Advanced Encryption Standard » (AES) [3] est l'algorithme qui illustrera l'application de la méthodologie de tests proposée dans le second chapitre sur ces modules. Nous détaillerons la mise en place des évaluations, les résultats expérimentaux, ainsi que leurs analyses dans les différentes sections dédiées.

Sommaire du chapitre

3.1 Procédure et matériel

Les implémentations des algorithmes cryptographiques dans les SoC sont tantôt logicielles tantôt matérielles. Les implémentations logicielles sont des programmes exploitant des librairies cryptographiques exécutés par un CPU tandis que les implémentations matérielles sont des coprocesseurs optimisés pour des calculs cryptographiques. Compte tenu du grand nombre d'algorithmes cryptographiques existants, nous avons choisi d'en sélectionner un qui nous servira de support d'étude. Il s'agit de l'« Advanced Encryption Standard » (AES) [3], un algorithme de chiffrement symétrique aujourd'hui très répandu et considéré comme sûr. Ainsi, dans ce chapitre, nous appliquons la méthodologie proposée section 2.1 pour évaluer la mise en place d'attaques matérielles sur des implémentations logicielles et matérielles de l'AES.

3.1.1 Véhicule de test

Les expérimentations ont été effectuées sur un unique véhicule de test. La principale raison est la limite de temps dont nous disposions pour mener cette étude. En effet, les SoC sont des systèmes complexes qui requièrent un temps d'étude non négligeable pour pouvoir développer des applications adaptées à nos expérimentations. C'est pourquoi nous avons sélectionné un unique système sur puce selon les critères suivants :

- Le modèle sélectionné est suffisamment représentatif du marché des objets complexes connectés.
- La documentation disponible à son sujet est détaillée et libre d'accès. Que ce soit avec le manuel utilisateur du fabricant ou des forums de discussions, il est relativement facile d'obtenir l'information recherchée tant que celle-ci n'est pas soumise au secret industriel.
- C'est un SoC protégé par un boîtier BGA en époxy (voir Annexe A.1) qui permet d'utiliser les champs électromagnétiques en tant que chemin d'attaque .
- Il intègre un unique CPU, ce qui permet d'estimer la complexité des manipulations pour un cas relativement simple compte tenu du fait que la plupart des SoC du marché sont souvent multicœurs.
- Il intègre un *cryptographic hardware accelerator* (C-HWA). Il s'agit d'un coprocesseur dédié à l'accélération matérielle des principaux algorithmes cryptographiques dont l'AES.

3.1.1.1 Caractéristiques principales du SoC

Le SoC sélectionné fait partie d'une famille de systèmes sur puce destinés à des applications multimédias. Ils sont très utilisés dans les secteurs de l'automotive et de l'IoT.

Comme expliqué dans la Section 1.1 décrivant les spécificités des SoC, nous étudions un système modulaire pourvu d'un large panel de fonctionnalités lui permettant d'exécuter les nombreuses tâches requises par les applications qu'il embarque. Le modèle retenu est

un SoC 32 bits gravé en technologie CMOS 40nm, embarquant un processeur ARM Cortex A9 MPCore^{TM}, ainsi que de nombreux modules IP dédiés au multimédia, à la sécurité, aux réseaux et à la gestion de sa consommation électrique. Son schéma bloc simplifié donne un aperçu de sa complexité (voir Fig. 3.1). Sur celui-ci, nous avons indiqué en orange les deux modules qui gèrent les implémentations de l'AES que nous étudions dans ce chapitre, à savoir le processeur principal et le coprocesseur cryptographique.

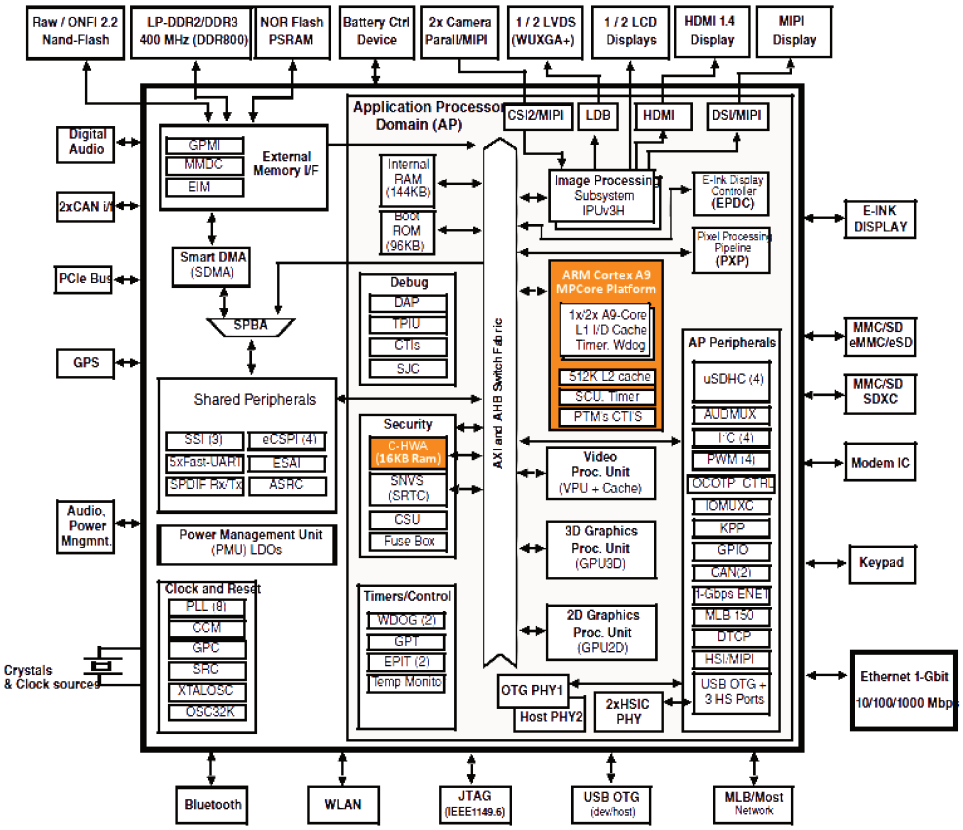


FIGURE 3.1 – Schéma bloc simplifié des différents modules composant le SoC retenu pour nos expérimentations

3.1.1.2 Le processeur principal

Le véhicule de test embarque un unique CPU de 32 bits dont la principale tâche est de gérer le système d'exploitation et les programmes applicatifs. C'est un processeur ARM Cortex A9 MPCore^{TM} avec deux niveaux de mémoire cache cadencés à une fréquence allant jusqu'à 1.3GHz. L'architecture de ce processeur est l'ARMv7-A. Des informations complémentaires concernant les processeurs ARM sont disponibles dans l'Annexe A.2.

3.1.1.3 Le coprocesseur cryptographique.

L'accélérateur cryptographique embarqué dans ce SoC est un coprocesseur qui regroupe des fonctions dédiées à la sécurité du SoC : accélération cryptographique, génération de nombres aléatoires et contrôle d'une mémoire RAM interne sécurisée. Ce module est implanté de la manière décrite dans le Ch. 1, Section 1.3.4. Il possède sa propre horloge et exécute ses tâches selon une machine d'états. C'est un module propriétaire soumis au secret industriel que nous étudions en « boîte-noire », autrement dit, sans informations fournies par le fabricant autres que celles disponibles pour les développeurs d'applications tierces. Parmi ces informations, il est indiqué, dans la documentation constructeur, que l'accélérateur matériel de l'AES bénéficie d'un dispositif de protection contre les attaques DPA. Il semblerait qu'il s'agisse d'un système de masquage des registres sans de plus amples de précisions.

3.1.1.4 La carte de développement

Afin de pouvoir solliciter le SoC avec les bancs d'expérimentation décrits dans le second chapitre, nous avons choisi une plateforme de développement adaptée. Comme le montre la figure 3.2, la plateforme retenue est un mini ordinateur qui permet d'utiliser une grande partie des périphériques proposés par le SoC (voir Fig. 3.1). Cette carte intègre notamment une mémoire RAM DDR3 de $1Go$ et un port microSD lui permettant d'exécuter le système d'exploitation Android préalablement chargé sur une carte.

Pour instrumenter nos manipulations, nous avons utilisé principalement le port série (J18), les GPIO (J13), le JTAG (J10) et une carte microSD (J7). Le port série est le canal de communication utilisé avec le Raspberry Pi. Le JTAG a servi durant la phase de développement des applications de test. La carte microSD contient les différents logiciels. Une partie d'entre eux sont nécessaires pour le démarrage et le chargement du système d'exploitation. L'autre partie est le système d'exploitation en question avec les applications dont nos programmes de tests pour l'AES.

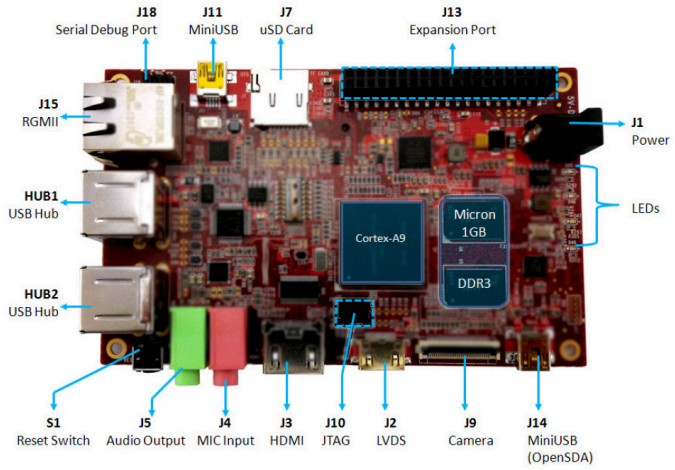
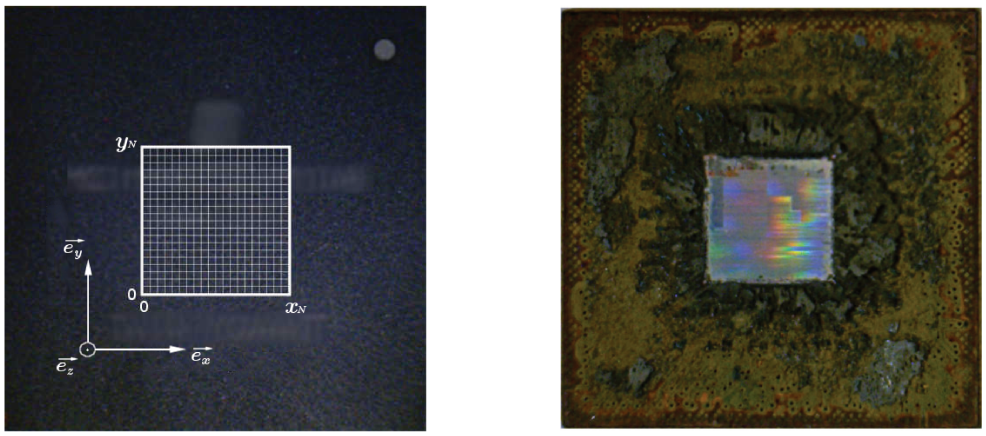


FIGURE 3.2 – Carte de développement, vue de dessus.

3.1.2 Paramètres side-channel et fault injection communs à toutes les expérimentations

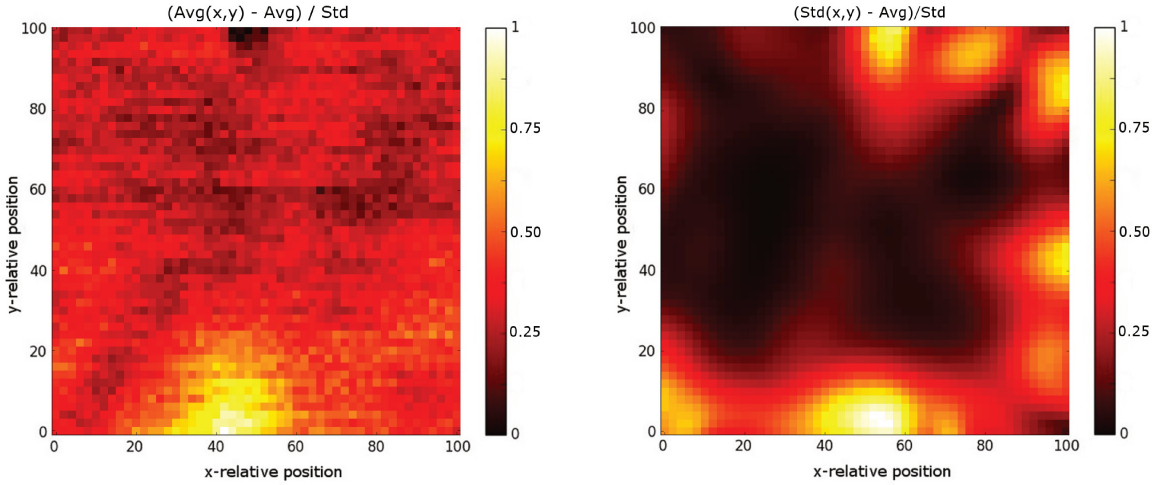
Les paramètres suivants sont communs à toutes les manipulations de ce chapitre.



(a) Boîtier du SoC avec les axes d'orientation

(b) Après traitement au bain d'acide nitrique fumant

FIGURE 3.3 – Zone spatiale considérée sur la surface du SoC



(a) Moyenne centrée réduite

(b) Ecart type centré réduit

FIGURE 3.4 – Amplitude des signaux EM mesurés sur la zone de recherche. Avg(x,y) représente la moyenne de l'amplitude d'un signal mesuré au point (x,y) et Avg représente la moyenne des signaux mesurés sur tous les points (x,y). De la même façon Std(x,y) et Std représentent respectivement l'écart type de l'amplitude des mesures au point spatial (x,y) et l'écart type global.

Localisation spatiale : Ne disposant pas de données suffisantes au sujet du *layout* du SoC, nous avons retiré le boîtier par voie humide (bain d'acide nitrique fumant) afin d'obtenir de plus amples informations. La figure 3.3-b montre la position du silicium dans le boîtier epoxy, le pourtour étant constitué de connexions. Le quadrillage de la figure 3.3-a représente la zone sur laquelle nous avons balayé la sonde et l'injecteur électromagnétique durant nos expérimentations. Cela représente une surface carrée de $6000\mu m \times 6000\mu m$.

Sonde de mesure : Toutes les mesures *side-channel* ont été faites avec une sonde Langer ICR HH-250. C'est un modèle avec un noyau torique de $\varnothing 250$ μm et une bande passante de 6GHz (Pour le détail des sondes voir (1), section 2.4 du Ch. 2). Pour chaque mesure, comme l'illustre la figure 3.5, la sonde est placée au plus près du boîtier du SoC ($z \to 0$).

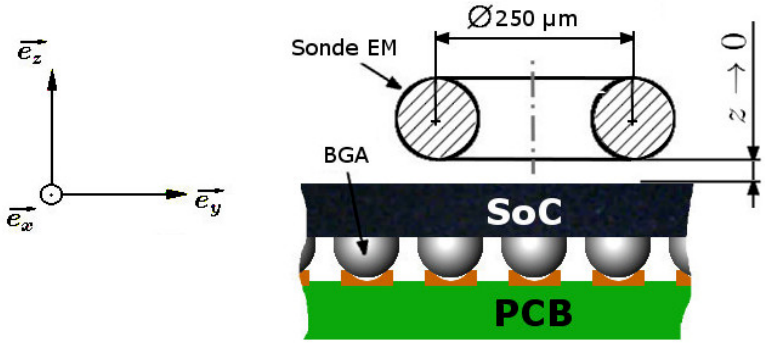


FIGURE 3.5 – Positionnement de la sonde de mesure électromagnétique au-dessus du boîtier du SoC

Fréquence et amplitude d'échantillonnage : Le processeur pouvant être cadencé jusqu'à 1.3GHz, l'échantillonnage de toutes les mesures électromagnétiques a été effectué à 5 GS/s. D'autre part, un balayage visant à déterminer l'amplitude des émissions électromagnétiques à la surface du SoC est présenté sur la figure 3.4. La dépendance spatiale de l'amplitude des rayonnements est à prendre en compte pour le calibrage de l'oscilloscope lors des mesures. Effectuer des mesures avec un calibre bien supérieur à l'amplitude maximale du signal observé en un point spatial ne permet pas d'obtenir un rapport signal-à-bruit suffisant pour exploiter les éventuelles informations contenues dans les mesures. A l'opposé, un calibre trop faible occasionne des signaux avec des zones de saturations. Dans les deux cas, cela rend difficile, voire impossible l'application d'attaques *side-channel*.

3.2 Faisabilité d'attaques matérielles d'un chiffrement AES effectué par le CPU

Dans cette partie, c'est le processeur principal du SoC qui exécute un programme de chiffrement sous le système d'exploitation Android. La méthodologie de test donnée section 2.1 est appliquée, à savoir une première phase d'analyse par canaux cachés (section 3.2.1), puis deux phases d'évaluation de la faisabilité d'attaques *side-channel* (section 3.2.2) et injection de faute (section 3.2.3).

L'implémentation logicielle de l'algorithme AES.

Pour pouvoir procéder à nos expérimentations, l'implémentation de l'algorithme de l'AES a été instrumentée avec l'ajout de portions de code spécifique et de signaux externes. Il s'agit d'un AES 128 bits en mode « Electronic Code Book » (ECB) sans contre-mesure. Les *S-Boxes* sont « codées en dur » et les données sont échangées à travers l'UART entre la carte de développement et le Raspberry Pi. Comme expliqué dans le paragraphe de localisation temporelle du signal (section 2.3.1.2, Ch. 2), des morceaux de codes sont ajoutés pour instrumenter nos manipulations. Un signal GPIO et des boucles *while* sont utilisés pour faciliter l'identification de l'AES. Afin de générer un motif facilement identifiable, plusieurs chiffrements AES sont exécutés entre la montée et la descente du GPIO. Le programme codé en C natif est résumé par le pseudo-code 3 donné en Annexe E. Dans celle-ci, on différencie deux séquences de boucles *while* : la première, contrairement à la seconde, inclut le chiffrement AES. Ceci est effectué afin d'aider dans la détection du signal de l'AES en comparant l'exécution des boucles *while* avec et sans la présence du calcul cryptographique.

3.2.1 Étude de la faisabilité d'une analyse simple de canaux cachés

Cette première phase de rétroconception est destinée à recueillir un maximum d'informations afin de pouvoir paramétrer les attaques par canaux cachés et par injection de faute. Cette première étape a été réalisée en utilisant le banc de mesure électromagnétique décrit dans la Section 2.4, Chapitre 2.

3.2.1.1 Mesures

La valeur de la clef est choisie aléatoirement et est fixée à $K =$ `0x3BE322662F3BE841502E794146052549`. Les messages M de 16 octets varient aléatoirement. Des séquences de quatre AES intercalés avec des boucles *while* sont jouées indéfiniment. Afin de diminuer l'effet de désynchronisation qu'apporteraient les modules de gestion dynamique de puissance et performances (DVFS, voir Annexe A.4), nous avons fixé le mode d'exécution du CPU à « `performances` » [35]. Ceci oblige ce dernier à travailler à fréquence maximale, sans chercher à optimiser sa consommation électrique, ce qui augmente l'amplitude des émissions électromagnétiques. Un premier balayage manuel est effectué sur la surface du SoC afin d'observer le type d'émissions observables. Plusieurs signaux caractéristiques ont ainsi été mesurés. La figure 3.6 présente les 4 points au-dessus desquels les signaux de la figure 3.7 ont été mesurés.

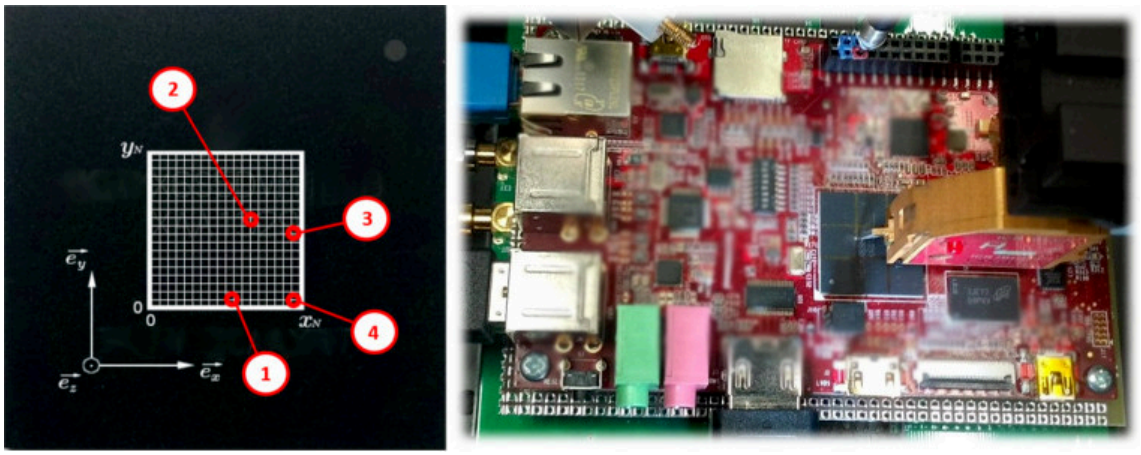


FIGURE 3.6 – Emplacements des mesures EM avec la sonde Langer ICR HH-250 pour des analyses préliminaires

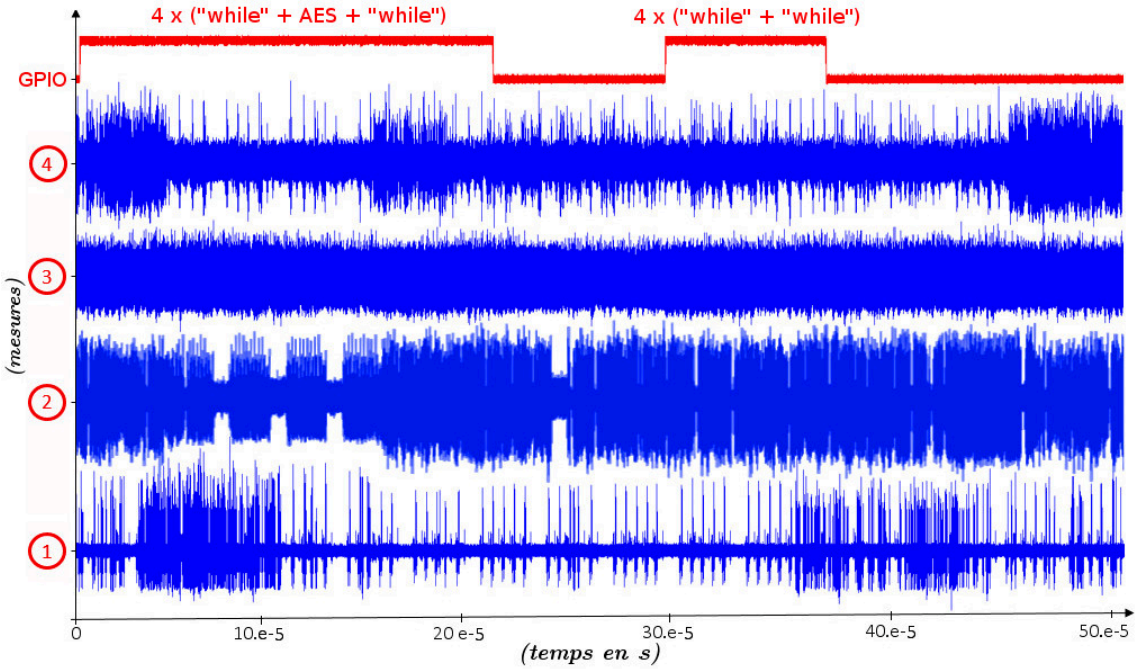


FIGURE 3.7 – Emissions EM mesurées aux points numérotés sur la figure 3.6

3.2.1.2 Analyses

Plusieurs informations essentielles sont obtenues avec ces premières mesures (voir Fig. 3.7) :

- Premièrement, on constate que la durée du GPIO au niveau "haut" est supérieure, dans le premier créneau, à celle du deuxième (200 μs contre 80 μs). Les deux parties contiennent des boucles *while* mais seule la première contient les 4 chiffrements AES.
- On observe une certaine similitude entre les signaux mesurés en (1) et (4) : une alternance de paquets et de pics très distincts. Il est probable que, dans ces deux zones, les bus soient reliés aux mêmes blocs IP et transportent le même type d'informations.
- La trace mesurée en (3), montre qu'avec cette sonde et la configuration des différents paramètres de mesure, aucune information apparente n'est visible à cet endroit.
- Enfin sur la trace (2) ($0 \leq t \leq 200\mu s$), on peut observer la répétition d'un même motif quatre fois.

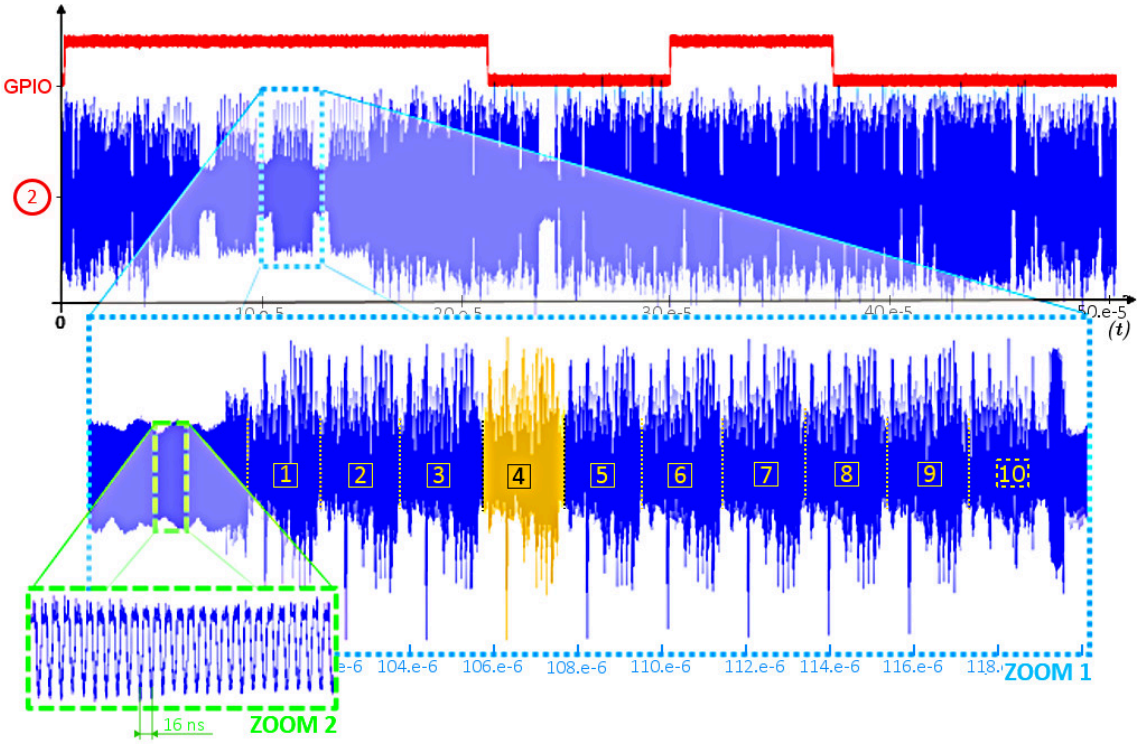


FIGURE 3.8 – Synthèse des informations apportées par des mesures sur la zone (2) de la figure 3.6

Cette dernière information établit de fortes présomptions quant à la possibilité d'observer les chiffrements AES dans la zone (2). Toutefois, le premier et le dernier motif apparaissent interrompus, ce qui signifierait que la première et la dernière boucle *while* ne sont pas visibles. De plus, les motifs des signaux mesurés pour $300 \leq t \leq 400\mu s$ ne permettent pas

de distinguer la répétition de processus qui identifieraient les boucles *while*. Ces dernières ont peut-être été estompées par les effets de la mémoire cache. Pour confirmer ou infirmer l'observation de l'AES, nous allons effectuer des mesures plus précises.

3.2.1.2.1 Mesures EM et analyses à l'emplacement n° 2. La figure 3.8 résume les différentes mesures qui ont été faites à l'emplacement (2). Des interruptions sont présentes sur les motifs des courbes acquises. Cela est certainement dû à l'aspect multi-processus des SoC. Cependant, les deuxième et troisième motifs de ce que nous supposons être l'AES sont tous intègres. C'est pourquoi nous avons focalisé nos mesures sur la zone repérée par le ZOOM 1. Cette zone montre que le motif n° [4] repéré en **jaune** se répète 10 fois pendant approximativement 18μs. De manière plus précise, ce motif est répété neuf fois avec un dernier motif légèrement différent des autres, comme les 10 tours de l'AES 128 bits avec le dernier sans l'opération `MixColumns`. Le ZOOM 2 présente une opération répétitive qui encadre les autres opérations. On identifie ainsi les boules *while*. Ces informations corroborent l'hypothèse d'observation de l'exécution du chiffrement AES par le CPU à cet emplacement.

3.2.1.3 Étude de la faisabilité d'une analyse simple de canaux cachés : conclusion.

Cette simple analyse *side-channel* nous a permis de localiser un emplacement du SoC au-dessus duquel l'activité du chiffrement AES par le CPU est observable par mesure électromagnétique. Les expérimentations montrent que cette phase est réalisable dans un laps de temps raisonnable, sans automatisation ni traitement des données. Seuls des « marqueurs » ajoutés dans le code sont utilisés pour repérer visuellement les motifs caractéristiques de l'AES. Cette analyse met également en avant le fait que l'exécution de l'AES peut être interrompue et la signature des rayonnements des boucles *while* amoindrie, voire supprimée. Nous supposons que cela est dû à l'aspect multitâche du système d'exploitation et aux mémoires cache présentes dans le SoC. Des expérimentations plus poussées permettraient de confirmer ces suppositions mais cela ne s'avère pas nécessaire dans le cadre de notre étude.

3.2.2 Étude de la faisabilité d'une attaque par analyse statistique de canaux cachés

Si la première phase d'analyse *side-channel* apporte des informations sur les zones au-dessus desquelles l'activité du chiffrement AES est mesurable, l'objectif de cette seconde phase est de mettre en évidence l'existence de fuites d'informations exploitables pour pouvoir appliquer une SCA.

3.2.2.1 Mise en place des manipulations

Les mesures électromagnétiques de cette section ont été effectuées au niveau de la zone mise en avant par l'étape précédente avec la même configuration. Afin de pallier les effets

des interruptions détectées, quatre chiffrements AES sont systématiquement joués et les acquisitions sont effectuées en se focalisant uniquement sur le second AES. La valeur de la clef est fixée à $K = \texttt{0x3BE322662F3BE841502E794146052549}$. Les messages M_i varient aléatoirement. En approximativement 3 heures, 12500 traces $T_i(t)$ ont été mesurées durant les chiffrements des M_i par K. Chaque trace est constituée de 125000 points temporels.

3.2.2.2 Utilisation du cœfficient de corrélation

Pour procéder à une analyse statistique, il est nécessaire au préalable de synchroniser temporellement les traces. Une d'entre elle est choisie comme référence et toutes les autres lui sont alignées suivant l'axe temporel. Ce procédé est effectué par un programme basé sur la méthode des moindres carrés qui aligne l'ensemble des traces sur une prise comme référence.

Afin de déterminer la faisabilité d'une attaque CPA, nous allons utiliser des cœfficients de corrélation sur les mesures effectuées (pour plus de détails, se reporter à l'Annexe B.4). Les traces mesurées sont regroupées dans des classes. Dans un premier temps, ce regroupement est effectué en fonction de la valeur du premier octet des messages utilisés durant les mesures. On note $M_{i,1} = v_1 \in \{0, \ldots, 255\}$, la valeur du premier octet du $i^{\text{ème}}$ message et T_i la trace mesurée durant l'utilisation de celui-ci. Nous calculons le cœfficent de corrélation pour chacune des 256 classes ainsi créées et en prenant en compte toutes les valeurs de message possibles. Autrement dit, pour chaque classe, on calcule 256 cœfficients de corrélation en considérant non pas v_1 comme valeur d'octet associée à la classe, mais $(v_1 \oplus p) \quad ; \quad p \in \{0, \ldots, 255\}$. Soit $H_{i,1,p} = H_W(v_1 \oplus p)$, le poids de Hamming la valeur associée. On calcule 256×256 cœfficients de corrélation linéaire $\Delta_{T,p}$ selon la formule B.5 rappelée en Annexe B.4 :

$$\Delta_{T,p}[t] = \frac{N \sum_{i=0}^{N-1} T_i[t]\ H_{i,1,p} - \sum_{i=0}^{N-1} T_i[t] \sum_{i=0}^{N-1} H_{i,1,p}}{\sqrt{N \sum_{i=0}^{N-1} T_i^2[t] - \left(\sum_{i=0}^{N-1} T_i[t]\right)^2} \sqrt{N \sum_{i=0}^{N-1} H_{i,1,p}^2 - \left(\sum_{i=0}^{N-1} H_{i,1,p}\right)^2}} \quad ; \tag{3.1}$$

On constate que l'on obtient des valeurs de corrélation maximales lorsque $p = 0$, *i.e.* lorsque la valeur de l'octet associé à une classe est la valeur correcte du message. La figure 3.9 illustre les 256 corrélations avec $p = 0$. Des maximums locaux représentés par des pics, sont présents en 3 endroits. A ces instants, il existe une relation linéaire entre la valeur du premier octet du message et les traces mesurées. Il s'agit d'une fuite d'informations liée au message.

On répète la même opération mais en considérant l'utilisation de la valeur du premier octet de la clef K_1. La valeur $(v_1 \oplus K_1 \oplus p)$ est considérée comme valeur associée à une classe. Ici aussi les corrélations sont maximales lorsque chaque classe est étiquetée avec sa valeur correcte *i.e.* lorsque $p = 0$. La figure 3.10 présente les 256 corrélations obtenues dans ce dernier cas. On observe la présence de deux pics représentant les instants temporels pour lesquels les données $(M_{i,1} \oplus K_1)$ sont manipulées. Ceci signifie que nous avons localisé des instants temporels pour lesquels la valeur de la mesure électromagnétique est linéairement corrélée à l'utilisation de la clef. La figure 3.11 détaille la discrimination possible entre les

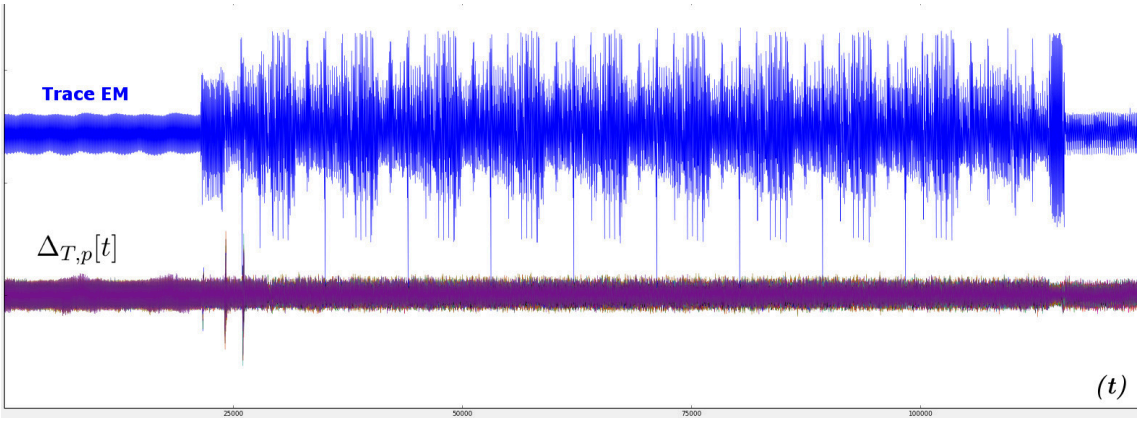


FIGURE 3.9 – Traces temporelles d'évolution du cœfficient de corrélation calculé pour les 256 valeurs du premier octet du message selon l'équation 3.1

différentes valeurs de corrélations. Celle-ci souligne le fait que le contraste observé entre les corrélations est relativement faible, ce qui est caractéristique de l'opération XOR. C'est pourquoi il est préférable de considérer une sortie d'opération *S-Box*, plus discriminante par sa non-linéarité [34]. Néanmoins, nous avons reproduit cette manipulation pour chaque octet de clef suivant, et des pics de corrélation similaires ont été observés avec des indices temporels légèrement supérieurs. Ceci s'explique par le fait que dans cette implémentation, les octets sont manipulés de manière séquentielle. Les différents calculs de corrélations ont révélé la présence de fuites d'informations liées aux valeurs d'octets de la clef. Des fuites d'informations sont présentes avec cette implémentation AES.

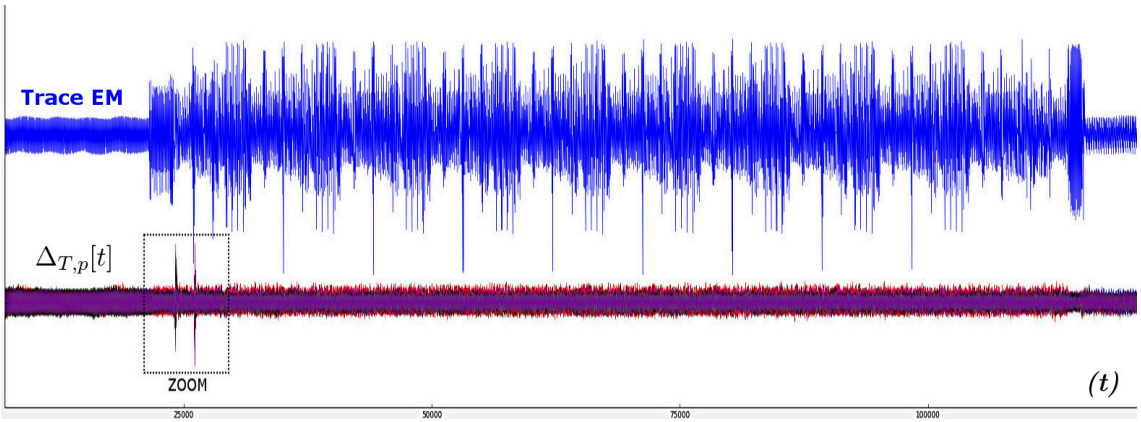


FIGURE 3.10 – Traces temporelles d'évolution du cœfficient de corrélation calculé pour les 256 valeurs de $(M_{i,1} \oplus K_1)$

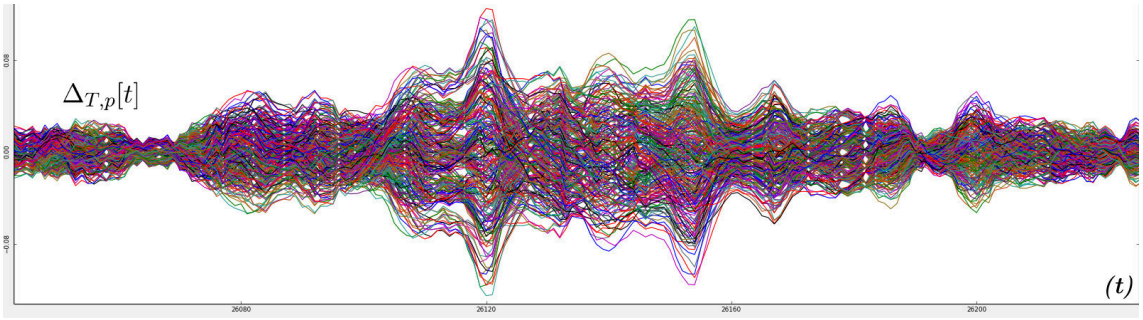


FIGURE 3.11 – ZOOM sur la figure 3.10 – Fuites d'informations relatives à la valeur du premier octet de la clef

3.2.2.3 Etude de la faisabilité d'une attaque par analyse statistique de canaux cachés : conclusion.

Nous avons montré que, pour une implantation cryptographique sans contre-mesure, le CPU produit des émissions électromagnétiques contenant des signaux caractéristiques des processus exécutés. Avec seulement 12500 traces, il est possible d'observer des fuites d'information relatives à la clef sur ce système. En augmentant le nombre de mesures, il est envisageable d'appliquer une CPA sur cette implémentation cryptographique et ainsi retrouver la valeur d'une clef secrète.

3.2.3 Étude de la faisabilité d'une attaque par injection de faute

Dans cette étape, nous cherchons à mettre en évidence la possibilité de créer des perturbations électromagnétiques durant le chiffrement AES du CPU, d'une part, pour évaluer le type d'erreurs que cela peut générer et, d'autre part, pour estimer la faisabilité d'une attaque DFA sur ces systèmes. Cette étape est réalisée en utilisant le banc d'injection électromagnétique décrit Section 2.5, Chapitre 2.

Il existe une multitude d'attaques DFA permettant de casser une implémentation AES par injection de faute. Une partie d'entre elles s'appuient sur les propriétés de l'opération `MixColumns` [50, 31, 92] et d'autres sont basées sur l'opération `KeySchedule` [60, 39, 136]. Nous avons choisi d'appliquer la méthode proposée par Dusart *et al.* [50] qui cible le début de la neuvième opération `MixColumns`. En effet, les auteurs expliquent que, durant cette phase, si un octet du `state` est changé en une valeur inconnue, il y aura une propagation de cette modification sur les quatre octets qui suivent l'opération `MixColumns`. Il existe ainsi une relation linéaire entre les quatre valeurs d'octets modifiées dans le chiffré. De cette manière, il possible de définir un ensemble d'hypothèses sur la valeur de l'octet à l'origine des modifications, ainsi que sur la dixième sous-clef utilisée. Cette méthode requiert un peu moins de 50 chiffrés erronés pour retrouver l'intégralité d'une clef d'AES 128 bits.

Les emplacements des chiffrés erronés exploitables pour cette attaque sont donnés dans le tableau 3.1.

	$C_{0,0}$	$C_{1,0}$	$C_{2,0}$	$C_{3,0}$	$C_{0,1}$	$C_{1,1}$	$C_{2,1}$	$C_{3,1}$	$C_{0,2}$	$C_{1,2}$	$C_{2,2}$	$C_{3,2}$	$C_{0,3}$	$C_{1,3}$	$C_{2,3}$	$C_{3,3}$
F_1	XX	E6	93	58	DE	C5	71	XX	28	2C	XX	B6	B8	XX	62	16
F_2	AF	XX	93	58	XX	C5	71	93	28	2C	2F	XX	B8	AB	XX	16
F_3	AF	E6	XX	58	DE	XX	71	93	XX	2C	2F	B6	B8	AB	62	XX
F_4	AF	E6	93	XX	DE	C5	XX	93	28	XX	2F	B6	XX	AB	62	16

Tableau 3.1 – Emplacement des erreurs dans les chiffrés pour que l'attaque DFA de [50] soit applicable sur notre configuration. Les $F_{1\leq i\leq 4}$ identifient le numéro de la colonne du `state` après l'opération `ShiftRows-9` dans laquelle la modification a été introduite. Les `'XX'` marquent les octets qui ont une valeur modifiée.

3.2.3.1 Mise en place des manipulations

Durant cette étape, la séquence AES entourée de *while* n'est jouée qu'une seule fois entre la montée et la descente du GPIO. Pour détecter l'obtention d'une faute durant le chiffrement, la clef et le message sont fixés à $K =$ `0x00112233445566778899AABBCCDDEEFF` et $M =$ `11111111111111111111111111111111`. Ainsi, en fonctionnement normal, la valeur du chiffré que l'on obtient est $C =$ `0xAFE69358DEC57193282C2FB6B8AB6216`. Avant chaque chiffrement, l'emplacement mémoire destiné à recevoir le chiffré est initialisé avec la valeur `0xCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC`. Ceci aide dans la caractérisation d'éventuelles fautes.

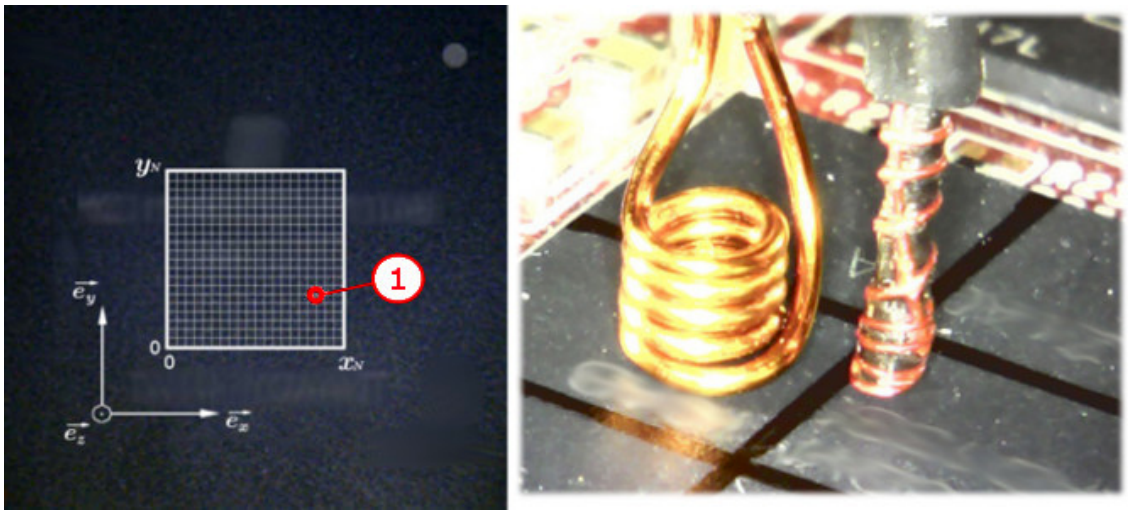


FIGURE 3.12 – Positionnement de l'injecteur EM.

3.2.3.1.1 Procédure d'injection de faute. Le choix de l'injecteur électromagnétique pour cette étape n'est pas simple. Plusieurs essais ont été réalisés avec différents modèles d'injec-

teurs (voir (2) Section 2.5, Ch.2). Nous présentons les résultats les plus significatifs obtenus avec l'injecteur électromagnétique cylindrique présenté sur la photo de droite de la figure 3.12. L'amplitude et la durée des impulsions électriques qui produisent de brusques variations électromagnétiques ont respectivement été réglées à $A_{inj} = +400V$ et $\Delta_{T_{inj}} = 6ns$. Grâce aux informations apportées par l'étape d'analyse *side-channel*, nous avons déterminé l'instant d'injection de faute, juste avant la moyenne des instants de l'exécution la neuvième opération `MixColumns` : $t_{inj} = 20,163\mu s$ avec t_0 pris à la montée du GPIO. On donne les deux définitions suivantes :

Définition 7. – `<mute>` **:** *on dit qu'un système est dans l'état* `<mute>` *lorsqu'il ne répond pas aux sollicitations qui lui sont faites sur ses ports I/O.*

Définition 8. – `<fauté>` **:** *on dit que le résultat d'une opération est* `<fauté>` *lorsque la valeur obtenue est différente de celle retournée durant le fonctionnement normal du système.*

Un premier balayage manuel sur la surface du SoC est effectué en générant des impulsions électromagnétiques à t_{inj} pendant le chiffrement AES. Aucun `<fauté>` n'est obtenu on constate seulement des blocages complets du système nécessitant des redémarrages. Le système est dans l'état `<mute>`. En procédant de la sorte, il est possible de localiser les endroits générant cet état et ainsi mettre en avant les zones sensibles aux perturbations. On suppose qu'en ajustant l'intensité des impulsions sur ces zones on peut faire fonctionner le système à la limite de sa défaillance et par conséquent induire des erreurs dans le processus exécuté qui est en l'occurrence un chiffrement. Cette limite de défaillance a été déterminée de manière empirique et consiste à régler l'intensité des impulsions électriques afin de n'obtenir que 10% de `<mute>` pour chaque point considéré. C'est de cette façon que la zone notée (1) sur la Fig. 3.12 a été sélectionnée pour procéder à la suite des expérimentations.

Après avoir placé l'injecteur sur la zone sensible et fixé les impulsions électromagnétiques pour que le système fonctionne à la limite de la défaillance, nous procédons à une campagne d'injection de perturbations afin de localiser l'instant produisant le maximum de `<fauté>` exploitables pour la DFA de Dusart *et al.* Celle-ci consiste en un balayage temporel sur une fenêtre d'une microseconde autour de t_{inj} avec un pas de $1ns$. Ainsi, pour chaque point temporel de la fenêtre $[\![t_{inj} - 500ns\ ;\ t_{inj} + 500ns[\![$, on répète 10 chiffrements AES durant lesquels on injecte une perturbation électromagnétique.

Les 10000 itérations ont été exécutées en approximativement 18 heures. Il s'est produit 1207 `<mutes>` et 412 `<fauté>`, soit respectivement $12,070\%$ et $4,120\%$ des chiffrements.

3.2.3.2 Analyse des fautes

Les 412 chiffrés erronés se répartissent selon 328 valeurs. La liste complète des chiffrés `<fauté>` obtenus durant cette manipulation est donnée en Annexe F.1. De nombreuses fautes difficilement explicables y sont listées. La figure 3.13 montre la répartition temporelle de celles-ci. Aucun instant ne semble privilégier la production d'erreurs dans le chiffrement. Nous nous sommes focalisés sur les fautes susceptibles de convenir pour appliquer une DFA selon Dussart *et al.* (voir tableau 3.1). Parmi l'ensemble des `<fauté>`, il a été identifié 45 candidats répartis selon 21 valeurs d'erreurs données dans le tableau 3.2.

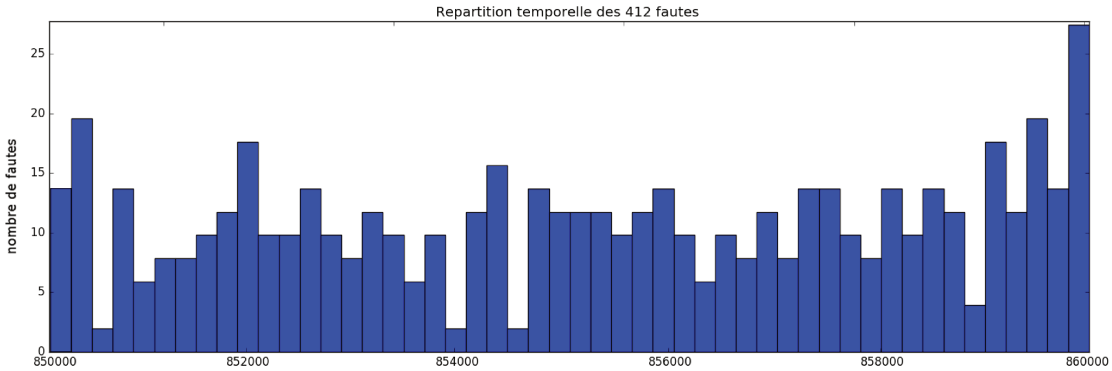


FIGURE 3.13 – Répartition temporelle de l'ensemble des chiffrés erronés sur la fenêtre de balayage temporel $[\![t_{inj} - 500ns \ ; \ t_{inj} + 500ns[\![$

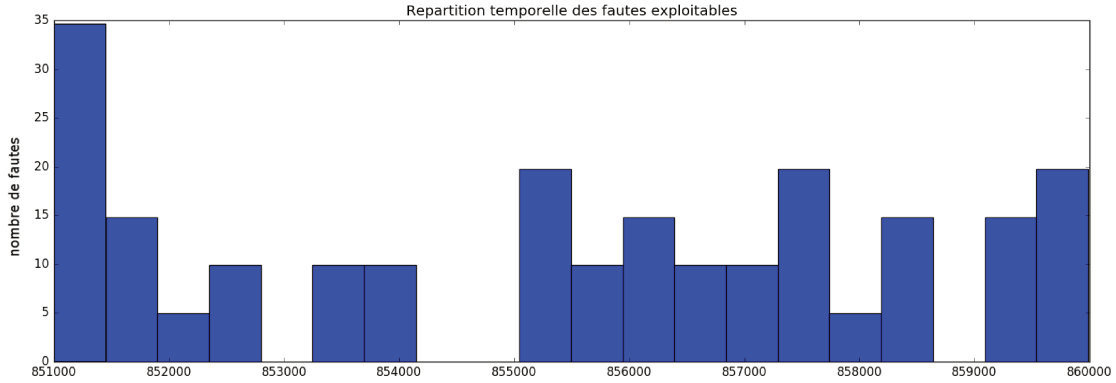


FIGURE 3.14 – Répartition temporelle des chiffrés exploitables sur la fenêtre de balayage temporel $[\![t_{inj} - 500ns \ ; \ t_{inj} + 500ns[\![$

De la même manière que pour l'ensemble des fautes, nous nous sommes intéressés à leur répartition temporelle (voir Fig. 3.14). Ici encore, malgré la présence de certains pics, aucun instant ne semble favoriser la génération de fautes exploitables. De plus, puisque le programme exécute chaque opération de manière séquentielle, on pourrait s'attendre à ce que les indices des octets modifiés soient ordonnées de façon similaire aux instants de perturbations électromagnétiques. L'expérimentation montre que les indices des colonnes de l'opération `ShiftRows-9` dans lesquelles les erreurs sont introduites n'ont pas de liens avec l'instant temporel d'injection.

Ces informations montrent l'incertitude temporelle qu'il peut y avoir lorsqu'on cible un programme exécuté par un processeur dans le contexte d'un système d'exploitation multi-tâche. En effet, l'ordonnanceur des tâches d'un OS tel qu'Android crée des interruptions inévitables lors de l'exécution d'un simple programme depuis le *userland* [8]. Ces interruptions génèrent du *jitter* dans les signaux qui désynchronisent l'instant de l'injection de la perturbation. Afin de détecter l'apparition d'une faute nous n'avons exécuté qu'un chiffrement AES. La désynchronisation de l'injection de fautes peut expliquer le fait que les fautes n'apparaissent pas de manière régulière temporellement.

n° identifant	Valeur du <fauté>																(occurences)	Type
(Chiffré Ref.)	AF	E6	93	58	DE	C5	71	93	28	2C	2F	B6	B8	AB	62	16		
0012	..	..	72	..	..	B7	..	..	70	..	..	..	..	..	..	8B	(1 times)	F_3
0014	..	..	..	1E	..	..	73	..	..	75	..	..	A3	..	..	..	(1 times)	F_4
0027	CE	..	..	..	..	..	..	78	..	..	78	..	..	9E	..	..	(3 times)	F_1
0044	..	..	..	F1	..	..	F1	..	..	29	..	..	68	..	..	..	(3 times)	F_4
0067	..	..	..	14	..	..	D1	..	..	49	..	..	A7	..	..	..	(2 times)	F_4
0075	02	..	..	..	..	..	..	88	..	..	A6	..	..	C0	..	..	(6 times)	F_1
0099	06	..	..	..	..	..	..	26	..	..	B6	..	..	8F	..	..	(2 times)	F_1
0161	..	..	..	1C	..	..	DB	..	..	6A	..	..	2A	..	..	..	(1 times)	F_4
0163	..	..	..	13	..	..	31	..	..	6D	..	..	6F	..	..	..	(5 times)	F_4
0171	16	..	..	..	..	..	..	66	..	..	EE	..	..	CA	..	..	(1 times)	F_1
0181	..	..	..	90	..	..	AF	..	..	79	..	..	2B	..	..	..	(5 times)	F_4
0198	..	07	..	..	E9	..	..	..	..	..	..	EE	..	..	83	..	(1 times)	F_2
0208	07	..	..	..	..	..	..	AC	..	..	AC	..	..	21	..	..	(1 times)	F_1
0220	..	C3	..	..	AD	..	..	..	..	..	..	3C	..	..	EF	..	(1 times)	F_2
0223	25	..	..	..	..	..	..	20	..	..	6E	..	..	1C	..	..	(1 times)	F_1
0224	1A	..	..	..	..	..	..	E4	..	..	A6	..	..	0D	..	..	(1 times)	F_1
0273	..	C3	..	..	EB	..	..	..	..	..	..	7C	..	..	99	..	(1 times)	F_2
0297	48	..	..	..	..	..	..	2C	..	..	2E	..	..	C3	..	..	(2 times)	F_1
0301	..	C3	..	..	E9	..	..	..	..	..	..	3C	..	..	40	..	(1 times)	F_2
0308	..	C6	..	..	E8	..	..	..	..	..	..	4E	..	..	A3	..	(1 times)	F_2
0316	..	..	5E	..	..	D1	..	..	7A	..	..	..	..	..	..	C3	(5 times)	F_3

Tableau 3.2 – 21 valeurs de 45 chiffrés erronés exploitables pour l'attaque DFA [50]. Les types de fautes $F_{1 \leq i \leq 4}$ font référence au tableau 3.1. Les "`..`" indiquent des valeurs d'octets non-modifiées.

3.2.3.3 Étude de la faisabilité d'une attaque par injection de faute : conclusion.

Cette étape a montré la faisabilité de perturber un processus cryptographique exécuté par un CPU. La perturbation de ce dernier n'a pas présenté de difficultés majeures et a pu être effectuée sans assistance automatisée ni traitement statistique. En seulement 10000 injections électromagnétiques, nous avons quasiment obtenu la moitié des chiffrés erronés nécessaires pour l'application de l'attaque DFA selon [50].

Malgré la difficulté de synchroniser l'instant d'injection de la faute avec le processus ciblé, les expérimentations ont montré la possibilité de générer les fautes escomptées. En revanche, cette incertitude temporelle est probablement en partie responsable des nombreuses fautes difficilement explicables qui ont été produites.

Une amélioration apportée à la précision du déclenchement de la perturbation permettrait de réduire ce type d'erreurs. Ceci pourrait s'envisager utilisant de la reconnaissance de motif en temps réel. Ainsi, l'impulsion électromagnétique serait déclenchée non pas au bout d'un délai fixé à partir du signal GPIO mais à partir d'un motif de référence dans le signal mesuré, synchronisé avec l'instant ciblé.

3.3 Faisabilité d'attaques matérielles d'un chiffrement AES effectué par l'accélérateur cryptographique

Dans cette partie, le calcul du chiffrement AES est effectué par le coprocesseur cryptographique du SoC. Nous allons soumettre ce dernier à une première phase d'analyse par canaux cachés (Section 3.3.1), puis à deux phases d'évaluation de la faisabilité d'effectuer des attaques *side-channel* (Section 3.3.2) et injection de faute (Section 3.3.3).

Utilisation de l'accélérateur matériel AES

Pour effectuer les mêmes chiffrements AES 128 bits en mode ECB avec le coprocesseur cryptographique du SoC, un programme *bare-metal* a dû être développé. Autrement dit, le programme utilisé pour la mise en place des tests est un programme auto-suffisant qui ne nécessite pas de système d'exploitation pour s'exécuter. La principale raison à cela est l'indisponibilité des *drivers* prenant en charge cet accélérateur dans les différents OS en accès libre. Ce programme est néanmoins semblable à celui de l'implémentation logicielle en C de la section 3.2. Un signal GPIO délimite l'exécution du chiffrement et des séquences d'instructions assembleurs `NOP` sont utilisées pour faciliter l'identification de l'accélérateur cryptographique dans les traces électromagnétiques. Le code est résumé par le pseudo-code 4 donné dans l'Annexe E. Dans celui-ci on a également deux parties conçues pour aider à l'identification du signal de l'AES : l'une avec le chiffrement AES espacé par des séquences de `NOP` et l'autre composée uniquement de `NOP`.

3.3.1 Étude de la faisabilité d'une analyse simple de canaux cachés

Dans un premier temps, nous avons procédé de la même manière que pour l'implémentation logicielle de l'AES en cherchant à identifier les différents types de signaux électromagnétiques observables.

3.3.1.1 Mesures

Puisque nous travaillons en mode *bare-metal*, il n'y a pas d'interruption de processus due à l'ordonnancement des tâches effectuées par le système d'exploitation, ce qui favorise les conditions de mesures. Par conséquent, un unique chiffrement AES est exécuté de manière répétitive avec des messages aléatoires M_i de 16 octets chiffrés par une clef de 16 octets $K =$ `0x3BE322662F3BE841502E794146052549`. Tous les IP-blocs non-configurés par notre code gardent leurs paramètres de configuration avec leurs valeurs par défaut. Nous nous sommes assurés que la fréquence de l'horloge du CPU exécutant notre code soit fixée à 792MHz. En revanche, aucune information précise sur la fréquence de fonctionnement de l'accélérateur cryptographique matériel n'est disponible.

Après avoir effectué un premier balayage manuel sur différentes zones à la surface du SoC, nous avons mesuré différents types de signaux. La figure 3.15 en présente quatre. Sur celle-ci, on peut noter que le temps d'exécution global est approximativement 50 fois plus court que celui de l'implémentation logicielle. Ceci s'explique par le fait que l'AES n'est joué qu'une fois, que le code est *bare-metal* avec des `'NOP'` à la place des *while* et surtout que le chiffrement AES est effectué par un accélérateur cryptographique matériel optimisé

dans ce sens. Quoi qu'il en soit, aucune information liée au fonctionnement de ce dernier n'a pu être mise en évidence, malgré la prise de différentes mesures, à divers emplacements et avec d'autres modèles de sondes.

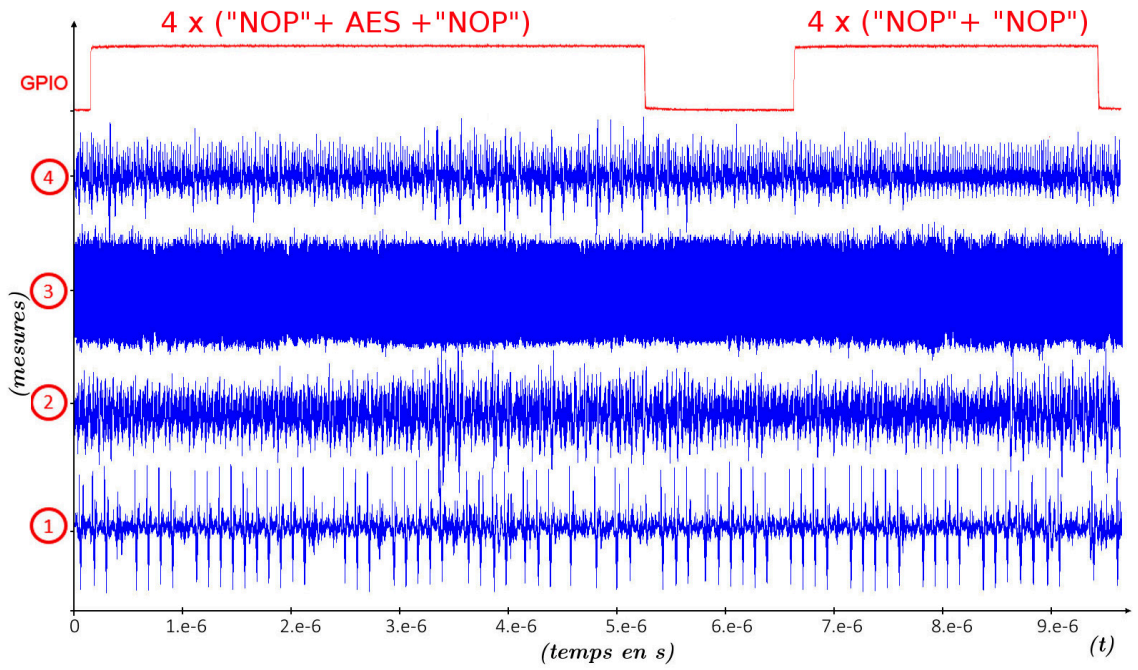


FIGURE 3.15 – Emissions électromagnétiques caractéristiques mesurées durant le balayage manuel à la surface du SoC

3.3.1.2 Nécessité d'automatiser les recherches

Etant donné que cette première approche n'a pas permis d'obtenir les informations nécessaires pour la mise en place des manipulations suivantes, nous avons dû procéder de manière plus minutieuse. Un unique chiffrement AES délimité par un signal GPIO et des `'NOP'` sont exécutés et une recherche automatisée plus fine avec des traitements statistiques est effectuée pour identifier des informations liées au fonctionnement de l'accélérateur cryptographique matériel.

3.3.1.2.1 Automatisation des mesures. En utilisant les fonctionnalités du banc électromagnétique décrit dans la section 2.4, nous avons effectué un ensemble de mesures des émissions électromagnétiques selon le quadrillage de figure 3.3-a, donnée en page 64. Le pas, dans les deux directions $(\vec{e_x}, \vec{e_y})$, est de $300\mu m$. Cela représente une matrice de mesures électromagnétiques de 21×21 points spatiaux. D'autre part, comme le montre la figure 3.4-b, page 64, il a fallu procéder à un asservissement du calibrage de l'oscilloscope afin d'acquérir des mesures d'une résolution maximale sans saturation *i.e.* avec le meilleur rapport signal-à-bruit possible.

3.3.1.2.2 Optimisation du rapport signal-à-bruit. Pour pallier la difficulté d'observation de signaux caractéristiques de l'accélérateur cryptographique, comme Longo *et al.* dans [85], nous avons utilisé le même principe que celui de la méthode des « Test Vector Leakage Assessment » (TVLA)[27]. Cette méthode préconise d'effectuer des mesures *side-channel* sur des calculs qui utilisent des ensembles de données choisis pour maximiser le rapport signal-à-bruit.

En l'occurrence, l'objectif est de détecter les zones spatiales pour lesquelles le chiffrement AES de l'accélérateur cryptographique émet des informations liées aux données manipulées. Autrement dit, nous recherchons les endroits où les émissions électromagnétiques traduisent l'utilisation d'une clef K, d'un message M et d'un chiffré C durant un laps de temps relativement court ($\leq 5\mu s$). Nous supposons que l'utilisation d'un modèle de fuite basé sur les distances de Hamming est adapté pour détecter le chargement ou le déchargement de valeurs dans les registres de l'accélérateur. Par conséquent, nous considérons des ensembles qui maximisent la distance de Hamming des données utilisées durant le chiffrement AES. Soit :

$$\mathcal{M} = \{M_0, M_{FF}\} \text{ , } \mathcal{K} = \{K_0, K_{FF}\} \text{ et } \mathcal{C} = \{C_0, C_{FF}\},$$

trois ensembles respectivement associés aux données du message, de la clef et du chiffré dans lesquels $X_0 = \sum_{i=0}^{i=15} \texttt{0x0}.16^i$ et $X_{FF} = \sum_{i=0}^{i=15} \texttt{0xFF}.16^i$, $X \in \{K, M, C\}$ sont deux valeurs de 16 octets ayant la plus grande distance de Hamming possible. On note de manière générique ces ensembles : $\mathcal{X} = \{X_0, X_{FF}\}$ avec $\mathcal{X} \in \{\mathcal{K}, \mathcal{M}, \mathcal{C}\}$ et $X \in \{K, M, C\}$.

Ainsi, pour chaque valeur appartenant à $\mathcal{X}$ et pour chaque point spatial $(x_i, y_j)_{0 \leq i,j \leq 20}$ de la surface du SoC, on mesure l'exécution de $n = 100$ chiffrements AES. Ceci représente l'acquisition de 441000 traces électromagnétiques effectuées en approximativement 12 heures pour chaque valeur de $\mathcal{X}$.

3.3.1.2.3 Traitement statistique. Des traitements statistiques sont appliqués afin d'identifier s'il existe une relation entre les mesures physiques et les données manipulées. Comme pour chaque application de ces traitements, il est nécessaire de synchroniser temporellement les traces mesurées. Ici encore, ce procédé est effectué par un programme basé sur la méthode des moindres carrés. Ensuite, pour chaque point spatial (x_i, y_j) et pour chaque ensemble $\mathcal{X} \in \{\mathcal{K}, \mathcal{M}, \mathcal{C}\}$, on applique les outils statistiques suivants aux différentes mesures :

$$S_{\mathcal{X}}(x_i, y_j, t) = \left(\frac{\overline{X_0(x_i, y_j, t)} - \overline{X_{FF}(x_i, y_j, t)}}{\sqrt{\frac{{\sigma_{X_0}}^2(x_i, y_j, t)}{n_{X_0}(x_i, y_j)} + \frac{{\sigma_{X_{FF}}}^2(x_i, y_j, t)}{n_{X_{FF}}(x_i, y_j)}}} \right)^2 \tag{3.2}$$

$$Thr\left(S_{\mathcal{X}}(x_i, y_j, t)\right) = \overline{S_{\mathcal{X}}(x_i, y_j, t)} + 3 \cdot \sigma_{S_{\mathcal{X}}}(x_i, y_j, t) \tag{3.3}$$

$$PoI_{S_{\mathcal{X}}}(x_i, y_j) = \{\forall t \ \mid \ S_{\mathcal{X}}(x_i, y_j, t) \geq Thr\left(S_{\mathcal{X}}(x_i, y_j, t)\right)\} \tag{3.4}$$

L'équation 3.2 est le SOST introduit Gierlichs *et al.* [59]. Il s'agit d'un outil statistique qui indique les instants durant lesquels des ensembles de mesures, construits selon des arrangements donnés, sont différents. Cet outil est utilisé lors d'analyses *side-channel* pour détecter les points d'intérêt. Ici, il est appliqué pour chaque point (x_i, y_j) en ne considérant que deux ensembles à la fois : X_0 et X_{FF}. Les valeurs $\overline{X_0(x_i, y_j, t)}$ et $\overline{X_{FF}(x_i, y_j, t)}$ sont les moyennes de ces ensembles. Le fait que le SOST soit normé par les écarts-types de chaque ensemble permet de le comparer avec les SOST calculés pour les autres points (x_i, y_j). La présence d'un pic dans cette grandeur indique l'instant auquel les deux ensembles de mesures électromagnétiques construits sur la valeur manipulée sont différents.

L'équation 3.3 est un seuil de confiance destiné à ne considérer que des différences significatives mises en avant par le SOST. Comme dans [59], on considère que les valeurs des données sont distribuées selon une loi normale, c'est pourquoi on utilise le seuil empirique « des trois sigmas » classiquement utilisé dans le traitement du signal. Seuls les instants durant lesquels le SOST est supérieur à ce seuil sont conservés, on considère qu'en-dessous, il s'agit de bruit.

L'équation 3.4 définit les échantillons temporels conservés. Ce sont les instants durant lesquels les ensembles construits sur les valeurs des données ont une différence significative. On suppose que ceci est dû à l'existence d'une relation entre le rayonnement électromagnétique et les données manipulées *i.e.* les PoI de cette analyse. La figure 3.16 illustre l'application de ces outils statistiques pour un point spatial (x_i, y_j).

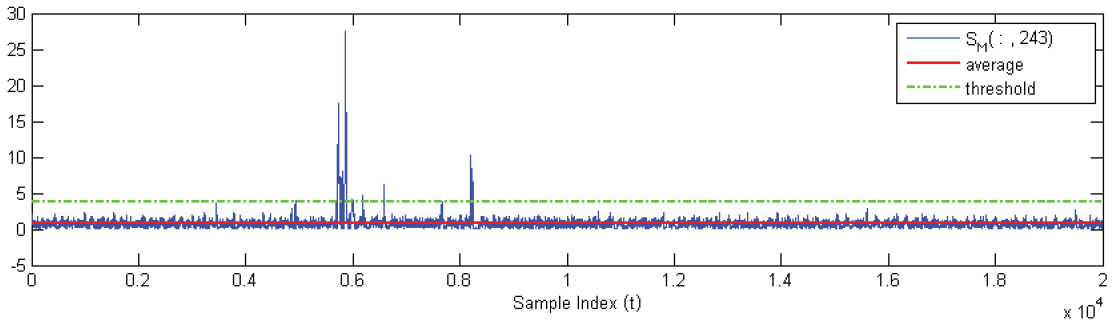
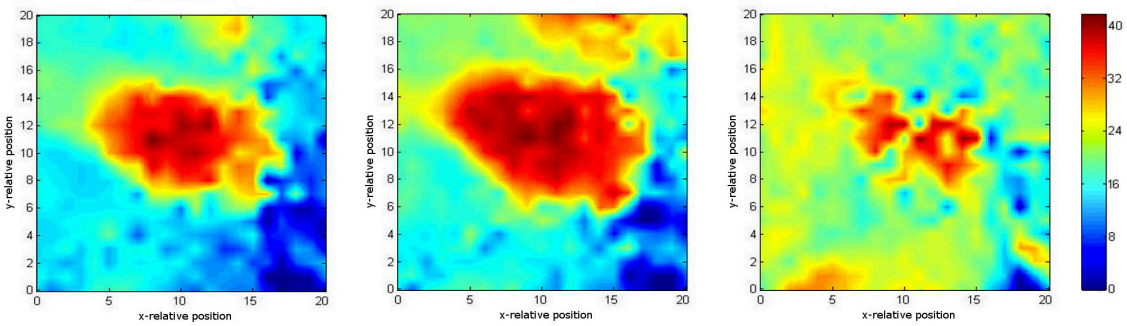


FIGURE 3.16 – Application de la fonction seuil définit par l'équation 3.3 sur un SOST calculé pour l'ensemble $\mathcal{M}$ au point (x_{11}, y_{12}). Seuls les points supérieurs à la valeur *threshold* $= Thr\left(S_{\mathcal{M}}(x_{11}, y_{12}, t)\right)$ sont conservés.

Pour interpréter au mieux ces résultats et avoir une cartographie en 2D de la surface du SoC, il faut exprimer l'information temporelle donnée par les PoI de l'équation 3.4 en un seul coefficient pour chaque point (x_i, y_j). Pour cela on définit les PoI spatiaux :

$$SPoI_{S_X}(x_i, y_j) = \sum_{t \in PoI_{S_X}} S_X(x_i, y_j, t) \tag{3.5}$$

En appliquant les équations 3.2, 3.3, 3.4 et 3.5 aux ensembles $\mathcal{K}$, $\mathcal{M}$ et $\mathcal{C}$, on obtient respectivement les cartographies a), b) et c) de la figure 3.17. Sur chacune d'elles, les zones en rouge traduisent des valeurs de $SPoI_{S_X}$ élevées.



a) $SPoI_{S_{\mathcal{K}}}(x_i, y_j)$ b) $SPoI_{S_{\mathcal{M}}}(x_i, y_j)$ c) $SPoI_{S_{\mathcal{C}}}(x_i, y_j)$

FIGURE 3.17 – Application des outils statistiques sur les signaux électromagnétiques mesurés pour les différents ensembles de données $\mathcal{K}, \mathcal{M}$ et $\mathcal{C}$.

Pour construire une cartographie qui synthétise la manipulation des données des ensembles $\mathcal{K}$, $\mathcal{M}$ et $\mathcal{C}$, on regroupe les trois cartographies en une seule. En premier lieu, chaque cartographie est normée par sa valeur maximale pour pouvoir être comparée aux autres. Puis, seuls les points spatiaux ayant une valeur supérieure ou égale à 0,9 sont conservés, *i.e* $\{(i,j) \in [\![0,20]\!]^2 \mid SPoI_{S_{\mathcal{X}}}(x_i, y_j) \geq 0,9\}$. Enfin, les trois cartographies sont additionnées point par point pour générer une représentation des zones donnant une information globale sur les manipulations des clefs $\mathcal{K}$, des messages $\mathcal{M}$ et des chiffrés $\mathcal{C}$. Cette dernière cartographie est donnée par la figure 3.18. Les zones claires sont celles pour lesquelles la métrique que nous avons utilisée révèle des coefficients de SOST simultanément élevés pour la clef, le message et le chiffré.

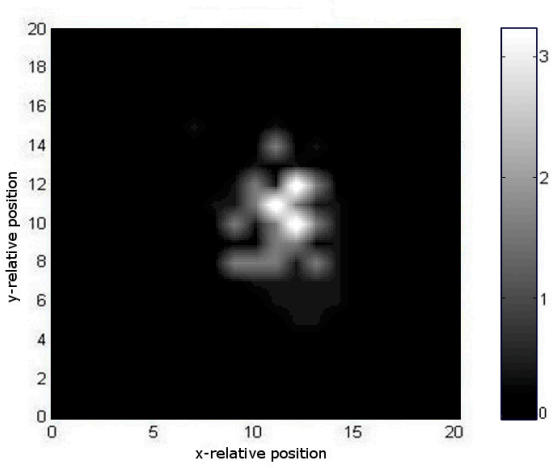


FIGURE 3.18 – Information globale obtenue par nos traitements statistiques à propos de la clé, du message et du chiffré

3.3.1.2.4 Mesure des signaux sur les zones révélées. Nous nous intéressons aux signaux émis au-dessus des zones révélées par les traitements statistiques précédents. Une mesure effectuée au point lumineux (x_{11}, y_{12}) de la Fig. 3.18 est présentée sur la figure 3.19. En comparant le signal électromagnétique des instants auxquels le GPIO est au niveau « haut », on remarque la présence d'un motif sur la première portion. Ceci conforte l'idée que nous observons une activité liée à l'accélérateur matériel, étant donné que le chiffrement n'est réalisé que dans la première partie. En effectuant la moyenne de 100 mesures électromagnétiques sur la zone délimitée par le ZOOM de la figure 3.19, on obtient le signal régulier $\overline{[T_{AES}]}_{100}$ de la figure 3.20. Celui-ci est composé de 10 répétions d'un motif pendant approximativement 300ns. On peut également observer un brusque pic sur la fin de ce signal, probablement dû à l'interruption générée par le C-HWA lorsque celui-ci indique au programme principal que le chiffré est disponible.

Avec ces dernières considérations et les différents traitements statistiques effectués, nous pouvons affirmer que nous avons localisé un emplacement au-dessus duquel il est possible d'observer l'activité de l'accélérateur cryptographique matériel.

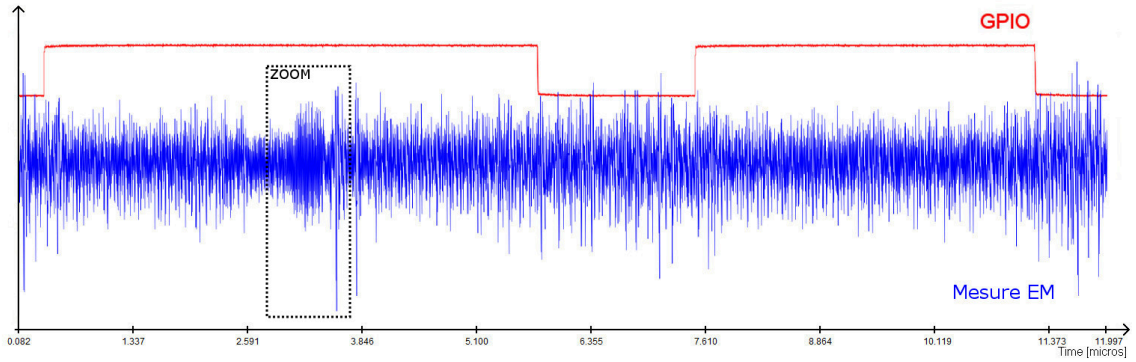


FIGURE 3.19 – Mesures électromagnétiques effectuées au point spatial (x_{11}, y_{12}) de la figure 3.18

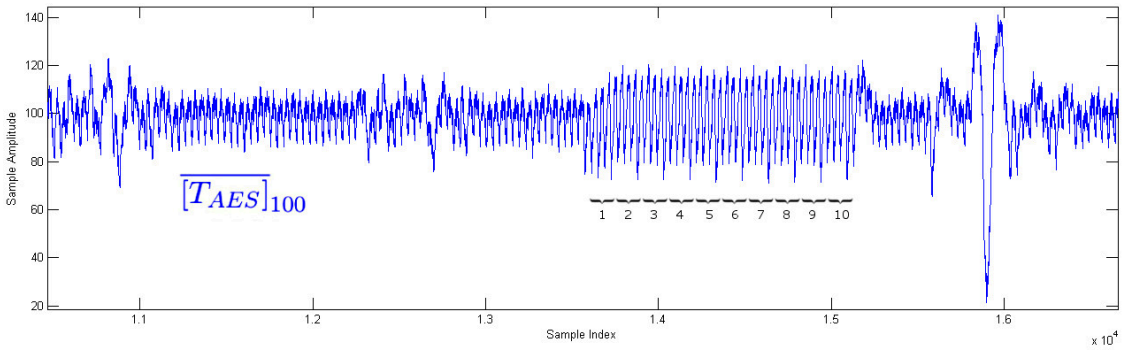


FIGURE 3.20 – Moyenne de 100 traces $\overline{[T_{AES}]}_{100}$ acquises durant la zone temporelle délimitée par le ZOOM sur la figure 3.19. Présence de motif répété 10 fois

3.3.1.3 Étude de la faisabilité d'une analyse simple de canaux cachés : conclusion

Cette analyse *side-channel* a présenté les difficultés pour mesurer l'activité d'un accélérateur cryptographique matériel intégré dans un SoC. Dans le cas présent, une simple analyse visuelle n'est pas suffisante. Le fait que l'accélérateur cryptographique soit optimisé peut laisser supposer que sa consommation et sa durée de calcul sont relativement plus faibles que celles d'autres bloc IP, comme le processeur principal par exemple. Ceci expliquerait la difficulté de sa localisation. En utilisant un code *bare-metal*, nous avons essayé de limiter le bruit généré par les autres IP-bloc. Cependant, comme pour [85], il a fallu recourir à des traitements statistiques. La synthèse des informations obtenues a finalement permis d'identifier des zones spatiales pour lesquelles les mesures électromagnétiques sont liées aux activités et aux données de l'accélérateur matériel cryptographique.

3.3.2 Étude de la faisabilité d'une attaque par analyse statistique de canaux cachés

Cette étape est destinée à évaluer la possibilité de mettre en place des attaques par analyse de canaux auxiliaires. Contrairement aux conditions de l'étape d'évaluation menée sur l'implémentation logicielle de l'AES (Section 3.2.2), l'implémentation matérielle évaluée dans cette section est protégée. En effet, le dispositif de masquage présent sur cet accélérateur est supposé restreindre l'efficacité des attaques par canaux cachés.

3.3.2.1 Mise en place des manipulations

En l'absence d'informations sur l'implémentation matérielle de l'AES, et sur celle des contre-mesures, l'évaluation de la faisabilité d'attaques se déroule en « boîte noire ». Il est donc nécessaire de recourir à des d'outils d'analyse statistique afin de localiser les instants susceptibles de produire des fuites d'informations exploitables *i.e.* les PoI. Parmi les divers outils statistiques utilisés pour la détection de fuites dans la cryptanalyse, nous en présentons trois qui sont couramment utilisés [42, 59, 37].

3.3.2.1.1 La détection de fuites. Les « détecteurs de fuites » ont le double objectif de déterminer les PoI et de mettre en avant l'existence de relations entre les mesures et les valeurs des donnés manipulées, *i.e.* les fuites d'informations. La présence de telles fuites suggère qu'il est envisageable d'appliquer des attaques *side-channel* au niveau des instants détectés. De manière complémentaire, chacun des outils présentés dans cette section considère un aspect particulier de la donnée. Pour utiliser ceux-ci, les traces mesurées T_i sont regroupées en fonction de la valeur de la donnée $H_{i,p}$ manipulée durant leur mesure. On note T_p l'ensemble des traces mesurées lorsque la donnée utilisée vaut H_p. Par exemple, si les fuites recherchées sont relatives à la valeur du premier octet du message, les traces mesurées sont classées en fonction de la valeur de l'octet utilisé pendant la mesure, soit selon 256 classes. Pour une série de chiffrements produisant N traces regroupées selon P classes, on considère les indicateurs de fuites suivants :

- Le SOSD (Sum of Squared Differences) :

$$sosd(T) = \sum_{p=0}^{P-1} n_p \left(\frac{\overline{T_p} - \overline{T}}{\sigma} \right)^2 \tag{3.6}$$

fournit la somme des écarts quadratiques des moyennes $\overline{T_p}$ de chaque classe par rapport à la moyenne de toutes les classes $\overline{T}$. L'écart de chaque classe est pondéré par le nombre de traces n_p qu'elle contient et est normé par l'écart type global de toutes les traces σ. C'est la somme des moyennes centrées pondérées par la population de chaque classe.

- Le SOST (Sum of Squared pairwise T-differences) :

$$sost(T) = \sum_{\substack{p>q \\ q=0}}^{P-1} \left(\frac{\overline{T_p} - \overline{T_q}}{\sqrt{\frac{{\sigma_p}^2}{n_p} + \frac{{\sigma_q}^2}{n_q}}} \right)^2 \tag{3.7}$$

il s'agit du SOST appliqué durant l'étape d'analyse *side-channel* précédente. Ici, il est appliqué sur les 256 classes et calcule la somme des écarts quadratiques entre les moyennes des classes deux-à-deux. Ces écarts sont normés avec σ_p et n_p, qui sont respectivement la variance et le nombre de traces de la p ième classe.

- Le cœfficient de corrélation linéaire ρ :

$$\rho(T) = \sum_{p=0}^{P-1} \frac{\mathrm{Cov}(T_p, H_p)}{\sigma_{T_p} \sigma_{H_p}} \tag{3.8}$$

utilisé par Brier *et al.* dans [34], il définit la qualité de la corrélation linéaire entre les traces T_p classées selon une valeur H_p considérée et les valeurs H_p elles-mêmes.

3.3.2.2 Application des outils statistiques

Les différents signaux présentés dans cette partie sont issus de mesures faites au-dessus des zones spatiales mises en avant par l'étape d'analyse *side-channel* de la section 3.3.1 précédente. Pour appliquer les outils de détection de fuites d'informations, deux jeux de 50000 traces chacun ont été construits. L'ensemble $M_R K_F$ a été mesuré durant les chiffrements de 50000 messages aléatoires de 16 octets par une clef fixe $K =$ `0x3BE322662F3BE841502E794146052549`, tandis que $M_F K_R$ a été mesuré pendant les 50000 chiffrements d'un message fixe $M =$ `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA` par des clefs de 16 octets qui varient aléatoirement. Pour être dans un état initial identique pour chaque mesure, la carte de développement a été systématiquement redémarrée avant celles-ci. Chacun des deux ensembles de 50000 traces a été acquis en approximativement 1 jour 6 heures.

Les résultats que nous présentons dans cette section sont issus de tests appliqués sur diverses classifications des traces de ces deux ensembles ($M_R K_F$ et $M_F K_R$). Une grande partie de notre travail a été de chercher à mettre en valeur des fuites d'informations liées au calcul de l'AES. Pour cela, différents regroupements des traces ont été effectués en

fonction des poids de Hamming de diverses tailles de données, en considérant la valeurs des octets ou des bits, avec des combinaisons $K \oplus M$, M^2, K^2, d'après les valeurs ou les poids de Hamming des octets d'opérations intermédiaires de chaque tour de l'AES, etc. Nous présentons ici les regroupements de traces qui ont donné les résultats les plus significatifs.

3.3.2.2.1 Rétro-ingénierie. Durant l'étape d'analyse par canaux cachés, les quatre traitements statistiques appliqués aux mesures ont permis de révéler les emplacements au-dessus desquels les émissions électromagnétiques sont liées aux activités de l'accélérateur cryptographique. Cependant, la cartographie 2D qui a ainsi été générée s'affranchit de la dimension temporelle des signaux. Or, pour appliquer des attaques *side-channel* nous avons besoin de cette information. Cette étape de rétroconception exploite l'aspect temporel des mesures afin de pouvoir définir des PoI et appliquer des attaques.

Chargement des registres de clef, message et chiffré. L'architecture du SoC ciblé dans cette étude est de type ARMv7. Par conséquent, les bus pour acheminer les données jusqu'au crypto-accélérateur ont une taille de 32 bits . Afin de détecter les instants de chargement de la clef, du message et du chiffré, les ensembles de traces mesurés ont été regroupés en fonction du poids de Hamming des 32 premiers bits des valeurs variables. Nous avons donc 33 valeurs de classes possibles. Plus formellement, pour une donnée D d'un processeur d'architecture N bits codée en binaire par $D = \sum_{i=0}^{N-1} d_i \cdot 2^i$, $d \in \{0,1\}$, on définit le poids de Hamming des bits compris entre les indices n et m ($n \leq m$) par :

$$H_W(D)_{n,m} = \sum_{i=n}^{m} d_i \qquad \text{avec} \qquad 0 \leq H_W(D)_{n,m} \leq (m-n+1) \tag{3.9}$$

Pour observer le chargement de la clef, du message et du chiffré par des bus de données de 32 bits, les 50000 traces acquises ont été successivement classées de la sorte :

- **Classement pour la clef** K : pour chaque trace T_i mesurée durant le chiffrement utilisant la donnée variable $K_i \in M_F K_R$, on associe le label $H_{i,p} = H_W(K_i)_{0,31}$
- **Classement pour le message** M : pour chaque trace T_i mesurée durant le chiffrement utilisant la donnée variable $M_i \in M_R K_F$, on associe le label $H_{i,p} = H_W(M_i)_{0,31}$
- **Classement pour le chiffré** C : pour chaque trace T_i mesurée durant le chiffrement utilisant la donnée variable $C_i \in M_R K_F$, on associe le label $H_{i,p} = H_W(C_i)_{0,31}$

Les Fig. 3.21, Fig. 3.22 et Fig. 3.23 présentent respectivement le calcul du SOSD, du SOST et du cœfficent de corrélation sur chacun de ces trois classements.
On constate premièrement que, dans ces circonstances, le SOST ne fournit pas d'informations exploitables. Aucun pic significatif ne se distingue réellement des autres. Deuxièmement, avec cette classification, la corrélation et le SOSD fournissent le même type d'information avec, toutefois, un RSB supérieur pour le second. Sur celui-ci on observe des pics à divers instants pour la clef, le message et le chiffré. Il semble que la clef soit chargée au niveau des 4 premiers pics rouges (Fig. 3.21), puis le message et pour finir le chiffré est récupéré au niveau du pic noir après le motif identifié comme étant l'interruption levée après

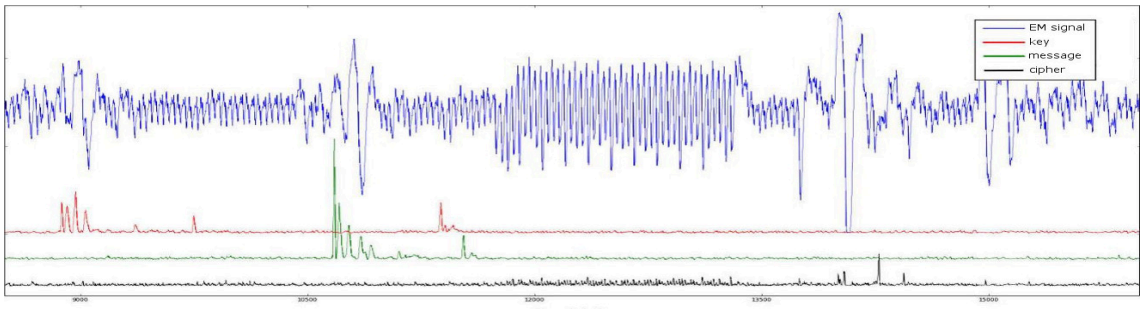


FIGURE 3.21 – SOSD calculés sur les poids de Hamming des 32 premiers octets des ensembles construits pour la détection du chargement des clefs, des messages et des chiffrés.

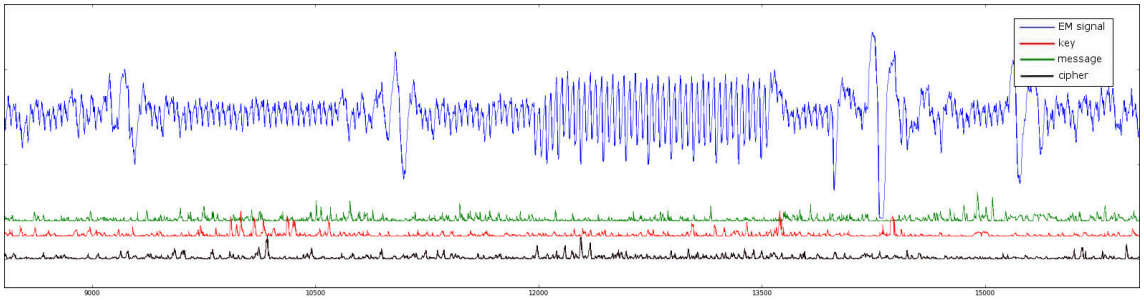


FIGURE 3.22 – SOST calculés sur les poids de Hamming des 32 premiers octets des ensembles construits pour la détection du chargement des clefs, des messages et des chiffrés.

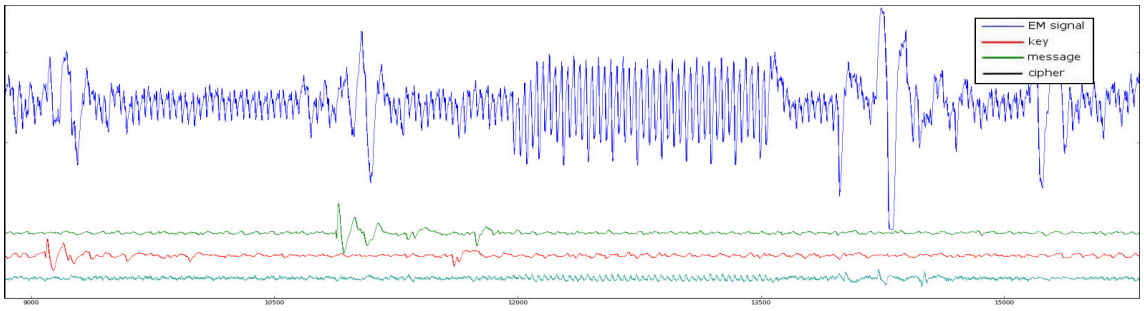


FIGURE 3.23 – Cœfficients de corrélarion calculés sur les poids de Hamming des 32 premiers octets des ensembles construits pour la détection du chargement des clefs, des messages et des chiffrés.

le calculs de l'AES. On observe également des pics liés à l'utilisation de la clé et du message après leur chargement, mais avant le chiffrement. L'information obtenue ici se limite au transport des données. D'autres tailles de données vont maintenant être considérées.

Taille des registres manipulant les données. Pour détecter si des registres de taille supérieure à 32 bits manipulent les données, on effectue des regroupements avec des valeurs du poids de Hamming calculées sur les tailles de données supérieures. Sur les figures qui suivent, **B**u **to B**v représentent une portion des octets du message, de la clef ou du chiffré. Ainsi, le poids de Hamming est calculé sur les octets allant du rang u jusqu'au rang v, soit avec l'équation 3.9 : $D \in \{K, M, C\}$, $H_W(D)_{8(u-1),(8v-1)}$.

Les trois outils statistiques ont été appliqués et seuls le SOSD et la corrélation fournissent des informations pour des poids de Hamming calculés sur des données de 32 et 64 bits. La Fig. 3.24 illustre le calcul du SOSD sur la clef. Plusieurs pics sont visibles sur celle-ci. Tout d'abord, nous allons nous intéresser au premier ensemble de pics identifié comme étant l'instant du chargement de la clef. Le ZOOM détaillé Fig.3.25 montre clairement la transition 32 à 64 bits. En effet, chaque pic de la période précédant la délimitation 32-64 bits représente la manipulation des données sous la forme de *mots* de 32 bits. On observe distinctement des pics temporellement décalés indiquant que les 128 bits de la clef sont chargés de manière successive par blocs de 32 bits. Passé cette délimitation 32-64 bits, les deux premiers et les deux derniers mots sont alignés entre eux. Ceci signifie qu'ils sont manipulés simultanément. Par conséquent, les registres de l'accélérateur cryptographique qui manipulent les données de la clef durant cette période sont de taille 64 bits. Cette information est corroborée par le fait que seuls des pics relatifs à des données de 64 bits sont observés après cette limite.

Des calculs équivalents ont été effectués pour le cœfficent de corrélation avec les mêmes regroupements de traces basées sur les poids de Hamming et des résultats similaires ont été obtenus. Cependant, un comportement particulier durant l'ensemble des tours de l'AES a été observé pour le chiffré lors de calculs effectués sur des traces T_i classées selon $H_W(C_i)_{0,63}$ et $H_W(C_i)_{64,127}$. La figure 3.26 présente les différentes corrélations obtenues. Sans informations sur l'implémentation matérielle, nous ne sommes pas en mesure d'expliquer simplement ce phénomène.

3.3.2.2.2 Fuite sur la valeur des données. Finalement, c'est le calcul du SOSD sur la valeur des données de chacun des 16 octets du message et de la clef qui a mis en avant une fuite d'information intéressante du point de vue cryptanalyse. En effet, pour les SOSD calculés sur la valeur des octets de rang 1, 6, 11 et 16 des messages ou des clefs, un pic est observable au même instant durant le début des tours de l'AES. Ces octets sont remarquables car ils sont situés sur la diagonale de la matrice d'état de l'AES. Le seul moment où la clef et le message sont utilisés simultanément pendant le chiffrement est durant l'opération `AddRoundKey`. Le fait que ce soit la diagonale du `state` de l'AES qui fuit, signifie que l'une des opérations du premier tour émet des rayonnements électromagnétiques en fonction de la valeur des octets manipulés aux rang 1, 6, 11, 16, et ce malgré la contre-mesure de masquage. Il est envisageable qu'une faille d'implémentation soit présente à cet endroit. La figure 3.27 présente le calcul du SOSD sur le $6^{\text{ème}}$ octet de la clef et du message.

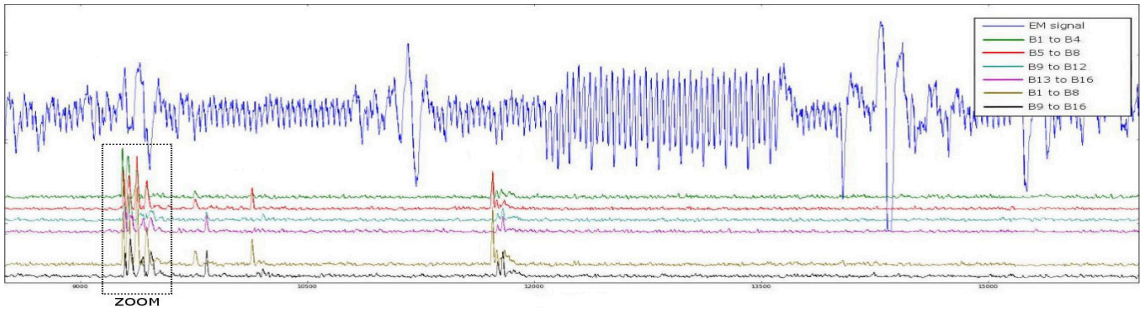


FIGURE 3.24 – Détection de la taille des registres manipulant la clef par le calcul du SOSD sur les poids de Hamming d'ensembles d'octets de 32 et 64 bits de la clef.

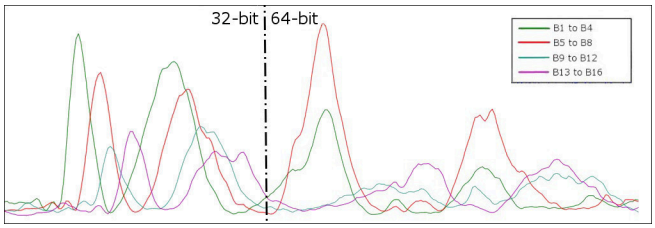


FIGURE 3.25 – ZOOM sur la figure 3.24 – Détail du changement de la taille des bus manipulant les données de la clef.

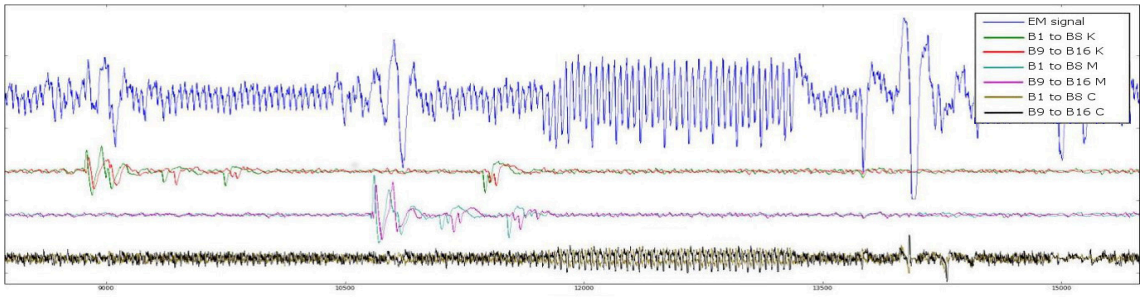


FIGURE 3.26 – Cœfficients de corrélation calculés en considérant les poids de Hamming d'ensembles d'octets de 32 et 64 bits de la clef, du message et du chiffré.

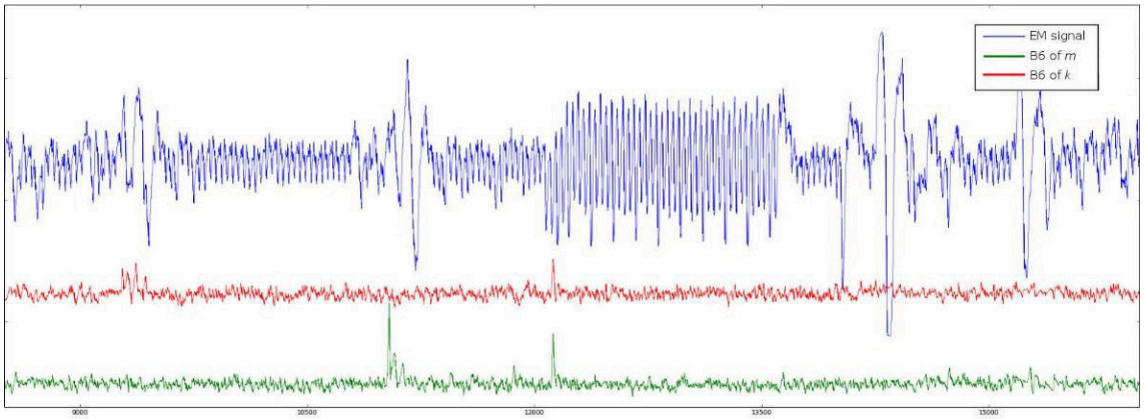


FIGURE 3.27 – SOSD calculés sur les des ensembles construits en considérant la valeur du $6^{\text{ème}}$ octet du message et de la clef de chiffrement.

Comparaison SOSD/SOST sur la valeur des données. Durant les diverses expérimentations manipulant les outils de détection de fuites, nous avons calculé le SOST et le SOSD sur le poids de Hamming et sur la valeur des octets. La figure 3.28 présente la comparaison des informations obtenues pour le $6^{\text{ème}}$ octet. Plusieurs constatations peuvent être données :

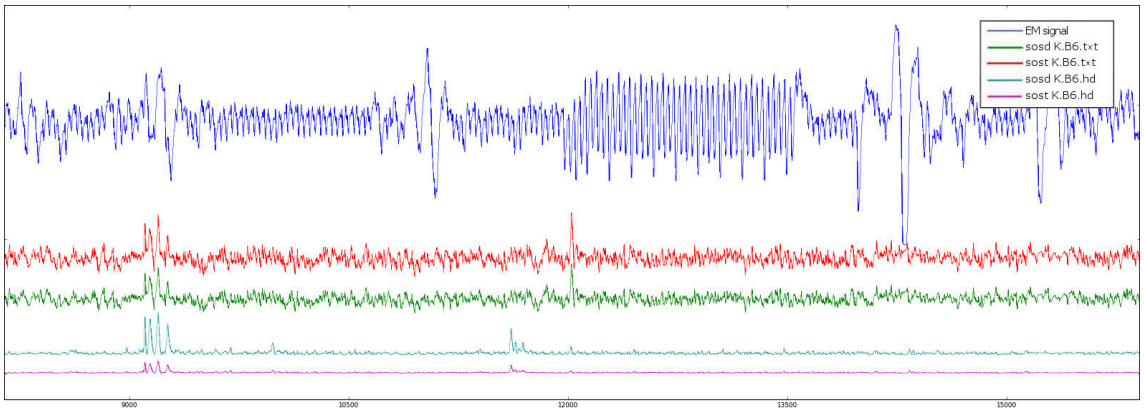


FIGURE 3.28 – Comparaison des SOSD, SOST calculés sur la valeur et le poids de Hamming du $6^{\text{ème}}$ octet de la clef de chiffrement

- Dans le cas d'un octet, ces deux outils statistiques fournissent la même information à un cœfficient multiplicatif près.
- Les calculs effectués sur des classes construites à partir des poids de Hamming des octets produisent des courbes plus « régulières » que lorsque qu'ils sont effectués directement sur leurs valeurs. Cela s'explique par la quantité de courbes présentes dans chaque classe. Lorsque l'on construit des classes suivant la valeur d'un octet, les traces sont rangées selon 256 valeurs alors qu'avec le poids de Hamming elles ne

le sont que selon 33 valeurs. Moins il y a de classes, plus la population de traces dans celles-ci est importante. Par conséquent, les moyennes utilisées dans les divers calculs de SOSD et SOST ont un meilleur RSB.

- Que l'on considère le poids de Hamming ou la valeur des octets, des informations relatives au chargement de la clef sont identifiables dans les deux cas.

- Lorsque c'est le poids de Hamming qui est utilisé, des fuites avant les tours de l'AES sont observables alors que celles pendant les tours ne le sont pas. Lorsque c'est la valeur des octets qui est utilisée, c'est l'inverse.

3.3.2.3 Attaque side-channel : conclusion

Cette section synthétise les informations que nous avons été capables d'extraire en « boîte noire ». L'utilisation de trois détecteurs de fuites a permis d'obtenir une multitude d'informations intéressantes pour un attaquant. Premièrement, les instants de chargement de la clef, du message et de récupération du chiffré ont été localisés. Ces opérations peuvent être une faille dans laquelle un scénario d'attaque peut-être envisagé. Des informations liées à l'architecture de l'accélérateur ont montré que celui-ci manipule les données dans des registres de taille supérieure à 32-bits. Il a également été possible de déterminer un instant temporel durant les tours de l'AES pour lequel des fuites d'informations relatives aux données manipulées sont identifiables. Ceci montre la présence d'une faille dans l'implémentation de l'AES matériel qui permet d'envisager d'approfondir les recherches dans ce sens.

L'absence de données sur l'implémentation matérielle est un obstacle pour la mise en place d'attaques matérielles, mais il ne s'agit pas d'une protection fiable car, avec seulement 50000 traces, il a été possible d'obtenir une grande quantité d'informations et de localiser certains PoI. Etant donné ces résultats, il est probable qu'en persévérant davantage, un attaquant puisse exploiter les fuites de cet accélérateur cryptographique.

3.3.3 Étude de la faisabilité d'une attaque par injection de faute

De la même manière que dans la Section 3.2.3 consacrée à la perturbation d'un chiffrement AES exécuté par un CPU, cette section cherche à évaluer la faisabilité de perturber le déroulement d'un chiffrement AES effectué par un accélérateur cryptographique matériel.

3.3.3.1 Mise en place des manipulations

Un unique chiffrement AES est effectué entre la montée et la descente du GPIO. Pour détecter l'obtention de fautes durant le chiffrement, les valeurs de la clef et du message sont fixées à K = `0x3BE322662F3BE841502E794146052549` et M = `0x00000000000000000000000000000000`. Ainsi pour un fonctionnement

normal de l'accélérateur cryptographique matériel, le chiffré obtenu est C = `0x524FF49CC3C5AE60B8A98156B1469E13`.

3.3.3.1.1 Procédure d'injection de faute. Etant donné que nous ne disposons pas du schéma de placement routage de la puce, nous ne pouvons pas anticiper sur la localisation des emplacements susceptibles de perturber l'accélérateur cryptographique lorsqu'ils sont soumis à des injections électromagnétiques. Pour cette raison, l'ensemble de la surface délimitant le silicium de la puce est balayé. Ainsi, les différentes injections permettront de mettre en évidence les zones au-dessus desquelles les perturbations électromagnétiques ont des effets sur le processus de chiffrement. Compte tenu des difficultés rencontrées lors de la procédure d'analyse *side-channel*, la procédure d'injection de fautes a entièrement été automatisée à l'aide des fonctionnalités du banc d'injection électromagnétique (voir Ch. 2, Section 2.5). Plusieurs campagnes d'injections avec les divers modèles d'injecteurs ont ainsi pu être effectuées.

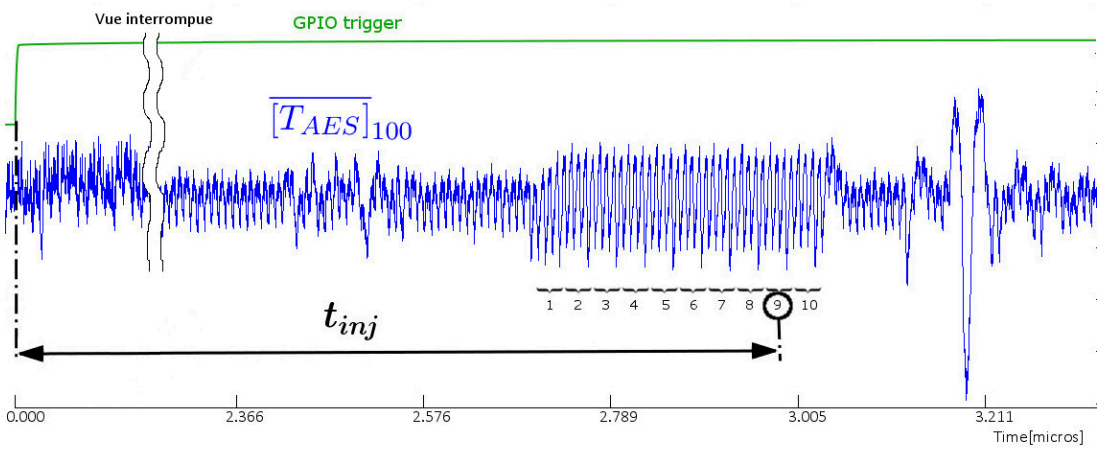


FIGURE 3.29 – Instant d'injection de la perturbation t_{inj} avec la montée du GPIO prise comme référence temporelle. $\overline{[T_{AES}]}_{100}$ est la moyenne de 100 traces de chiffrement AES.

Dans les différents balayages, on considère une surface rectangulaire de longueur L et de largeur l en μm. L'ensemble de la zone quadrillée représenté sur la figure 3.3-a, page 64, est un carré avec $L = l = 6000\mu m$. On définit Δp_x et Δp_y (en μm) les pas entre chaque point pour les directions respectives $\vec{e_x}$ et $\vec{e_y}$. Cela représente une matrice de mesure de $(\frac{l}{\Delta p_x}+1)(\frac{L}{\Delta p_y}+1)$ points spatiaux. Pour chaque point, N chiffrements AES sont répétés, durant lesquels des perturbations électromagnétiques sont générées. Au final, cela représente $N \cdot (\frac{l}{\Delta p_x}+1)(\frac{L}{\Delta p_y}+1)$ chiffrements par cartographies pour lesquels autant de perturbations électromagnétiques sont injectées. Ce sont des manipulations dont la durée dépend des pas Δp_x,Δp_y et du nombre d'essais par point N, mais aussi du nombre de `<mute>` générés (voir la défintion 7). Elles peuvent rapidement devenir très chronophages car, à chaque `<mute>`, il faut redémarrer le système et ceci dure approximativement 2 secondes de plus que lorsque le procédé se déroule normalement. C'est pourquoi il a fallu

trouver un compromis entre tous ces paramètres pour chaque cartographie, afin d'obtenir suffisamment d'informations sur la perturbation électromagnétique et ce, en un temps raisonnable.

Comme pour l'étude sur le CPU, nous cherchons à appliquer la méthode DFA de Dussart *et al.* [50]. Ceci implique d'introduire une perturbation juste avant la neuvième opération `MixColumns` du chiffrement AES. Grâce aux étapes *side-channel*, nous sommes en possession d'une information temporelle précise quant au déroulement de cette opération. La montée du GPIO est prise comme référence et l'injection de la perturbation électromagnétique est effectuée à l'instant $t_{inj} = 2,988\mu s$ (voir Fig. 3.29). Pendant chaque cartographie, les impulsions électriques du générateur ont des amplitudes et des durées fixes $(A_{inj}, \Delta_{T_{inj}})$.

3.3.3.2 Résultats expérimentaux

Sur les nombreux essais effectués avec les différents injecteurs à notre disposition pour cette étude (voir Fig. 2.9 Ch. 2), nous présentons ici les résultats les plus significatifs. Ceux-ci on été obtenus avec les trois injecteurs représentés sur la figure 3.30. Chacun d'entre eux a balayé la globalité de la surface quadrillée représentant la position de la puce dans le boîtier du SoC.

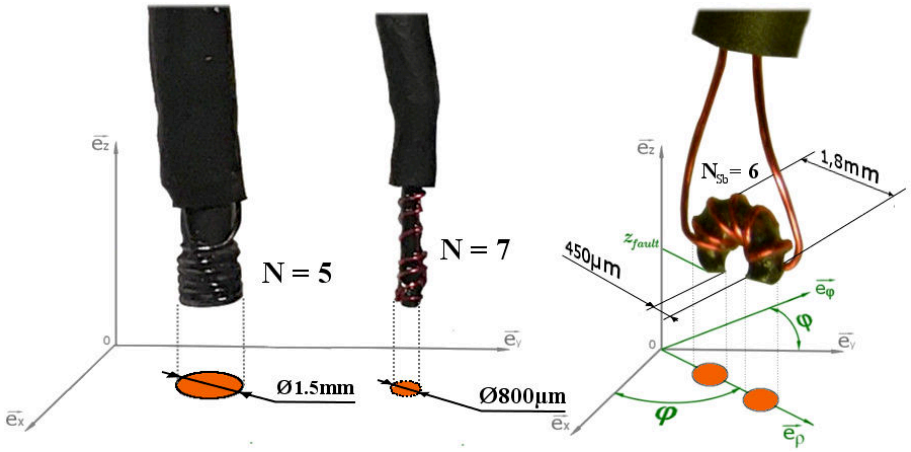


FIGURE 3.30 – Injecteurs EM produisant les résultats les plus significatifs sur cet accélérateur cryptographique

Les expérimentations ainsi que les analyses effectuées avec ces trois injecteurs sont décrites dans les sections respectives :

- Injecteur EM Cyl.-∅1500-N5 : Section 3.3.3.2.1
- Injecteur EM Cyl.-∅800-N7 : Section 3.3.3.2.2
- Injecteur EM Ω-∅1800.450-N6 : Section 3.3.3.2.3

3.3.3.2.1 Perturbations avec l'injecteur EM Cyl.-∅1500-N5

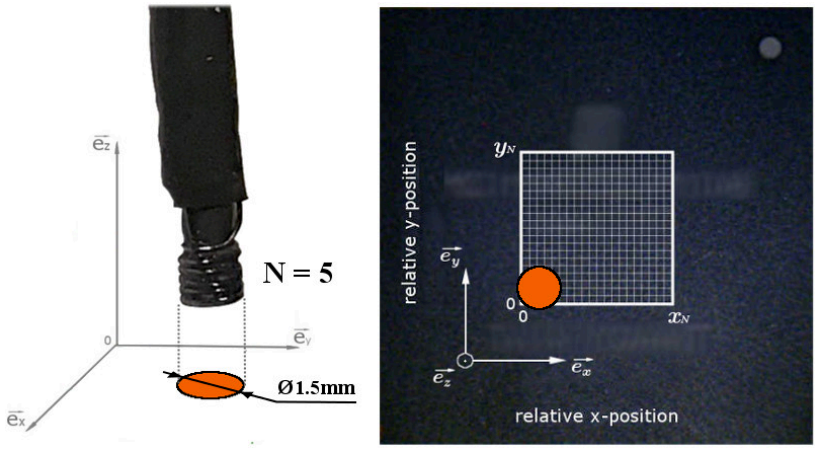


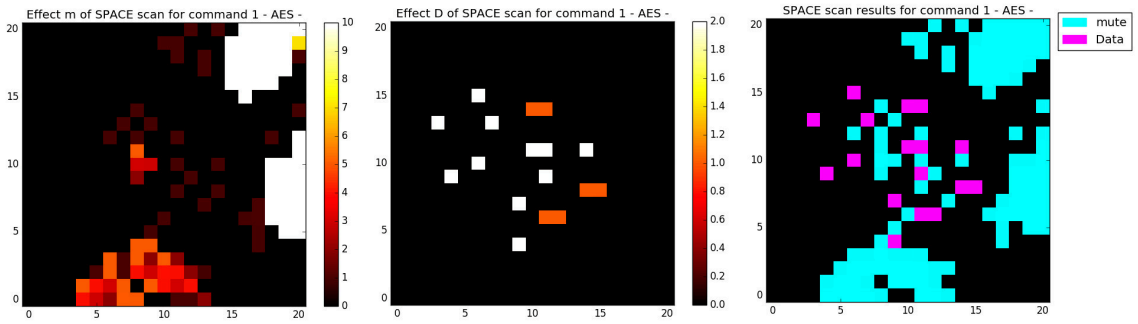
FIGURE 3.31 – Injecteur Cyl.-∅1500-N5. La superficie occupée par cet injecteur au-dessus du SoC est la surface orange.

Cet injecteur est présenté figure 3.31. Comme tous les injecteurs à notre disposition, ce modèle a été fabriqué à la main. Il a un diamètre approximatif de 1,5 mm autour duquel sont enroulées $N_{Sb} = 5$ spires de cuivre. Ce premier modèle a été choisi pour sa taille et sa symétrie axiale autour de l'axe $\vec{e_z}$. En effet, sans connaissance préalable de la disposition des différents block-IP dans la puce, un premier balayage permettra de localiser les zones sensibles aux impulsions électromagnétiques. Les paramètres du balayage sont donnés dans le tableau 3.3.

Paramètre	**Valeur**	
Zone balayée : $L = l$	6000 μm	} Matrice (21×21) ; 4410 essais
Pas : $\Delta p_x = \Delta p_y$	300 μm	
Nombre d'essais par point : N	10	
Amplitude de l'impulsion : A_{inj}	$+400V$	
Durée de l'impulsion : $\Delta_{T_{inj}}$	$6ns$	

Tableau 3.3 – Paramètres de balayage avec l'injecteur EM Cyl.-∅1500-N5

Les $36.10^6 \mu m^2$ du SoC ont été balayés en 96 minutes. Sur les 4410 injections électromagnétiques, il y a eu 600 `<mute>` et 14 chiffrés `<fauté>` soit respectivement $13,605\%$ et $0,317\%$ des chiffrements. La figure 3.32 résume ces résultats en y ajoutant une information spatiale. Sur ces cartographies, on remarque que les `<mute>` définissent deux zones très sensibles au-dessus desquelles, lorsqu'on produit 10 injections électromagnétiques, on obtient 10 blocages du système. Les `<fauté>` se répartissent au centre de la puce et ne sont pas plus de 2 par point spatial.



Nombre de `<mute>` Nombre de `<fauté>` `<mute>`/`<fauté>`

FIGURE 3.32 – Cartographies synthétisant les comportements observés à la surface du SoC lors du balayage avec l'injecteur EM Cyl.-∅1500-N5

n° identifant	Valeur du `<fauté>`	(occurences)
(Chiffré Ref.)	`52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13`	
0000	`.. .. .. .. .. .. .. E0 .. .. .. D6 .. .. .. ..`	(1 fois)
0001	`.. .. .. .. .. .. .. 10 .. .. .. .. .. .. .. ..`	(6 fois)
0004	`70 84 F6 D7 A1 A5 71 74 15 EA 46 6C 75 C1 DC 94`	(1 fois)
0005	`07 77 6F CA E6 A8 69 82 CE 37 B6 B1 5E 23 00 6B`	(1 fois)
0006	`00 00 .. .. 70 01 .. .. 00 00 .. .. 00 00 .. ..`	(1 fois)
0007	`D8 E6 D4 F1 74 C4 45 CF 9A 7B FF 08 B3 D8 4F 9C`	(1 fois)

Tableau 3.4 – Extrait des chiffrés erronés pour l'injecteur EM Cyl.-∅1500-N5. Les `".."` indiquent des valeurs d'octets non-modifiées.

Analyse des fautes. On dénombre 9 valeurs de chiffrés erronés. Le tableau 3.4 donne uniquement celles mentionnées dans cette analyse et la liste intégrale est présentée dans l'Annexe F.2.1. Pour chercher à analyser l'origine de la faute nous avons développé un logiciel simple basé sur un déchiffrement AES : `reverse_AES.py`. Ce programme prend en entrée la valeur du chiffré `<fauté>` et la clef de chiffrement, puis retourne les valeurs du *state* de l'AES à l'issue de chaque opérations intermédiaires. Ceci permet d'identifier les opérations pour lesquelles une perturbation a probablement pu causer l'obtention de la valeur du `<fauté>`. L'utilisation de ce programme pour le `<fauté>` n° 0000 est donnée figure 3.33. Sur celle-ci on observe que plusieurs modifications d'octets dans les opérations successives de l'AES peuvent produire le résultat fauté. Si l'on observe les emplacements des octets impliqués dans ces opérations, deux modifications sont les plus probables. Il s'agit des modifications se produisant à l'issue des opérations `MixColumns-9` (`MC9`) et `ShitRows-9` (`SR9`). En effet, pour ces deux opérations, on peut expliquer la production du `<fauté>` n° 0000, soit par la modification de deux octets mitoyens dans la matrice du *state* dans le cas de l'opération `MC9`, soit par la modification des deux premières colonnes du *state* dans le cas de l'opération `SR9`. Les modifications des octets dans les opérations qui précédent `SR9` sont moins évidentes à expliquer.

Valeur du <fauté> n° 0000																
Chiffré	52	4F	F4	9C	C3	C5	AE	60	B8	A9	81	56	B1	46	9E	13
<fauté>	..	..	..	..	..	..	..	E0	..	..	..	D6	..	..	..	..
10ème Tour																
ARK10	52	4F	F4	9C	C3	C5	AE	60	B8	A9	81	56	B1	46	9E	13
<fauté>	52	4F	F4	9C	C3	C5	AE	E0	B8	A9	81	D6	B1	46	9E	13
(RK10)	43	F0	00	B0	FE	D7	11	00	07	38	C5	5B	62	86	73	79
SR10	11	BF	F4	2C	3D	12	BF	60	BF	91	44	0D	D3	C0	ED	6A
<fauté>	11	BF	F4	2C	3D	12	BF	E0	BF	91	44	8D	D3	C0	ED	6A
SB10	11	C0	44	60	3D	BF	ED	0D	BF	12	F4	6A	D3	91	BF	2C
<fauté>	11	C0	44	E0	3D	BF	ED	8D	BF	12	F4	6A	D3	91	BF	2C
9ème Tour																
ARK9	E3	1F	86	90	8B	F4	53	F3	F4	39	BA	58	A9	AC	F4	42
<fauté>	E3	1F	86	A0	8B	F4	53	B4	F4	39	BA	58	A9	AC	F4	42
(RK9)	DB	BE	93	FD	BD	27	11	B0	F9	EF	D4	5B	65	BE	B6	22
MC9	38	A1	15	6D	36	D3	42	43	0D	D6	6E	03	CC	12	42	60
<fauté>	38	A1	15	5D	36	D3	42	04	0D	D6	6E	03	CC	12	42	60
SR9	0D	67	F9	72	A9	19	0E	5A	46	B5	52	17	38	3C	55	AD
<fauté>	A6	0C	32	49	E0	57	C9	DD	46	B5	52	17	38	3C	55	AD
SB9	0D	3C	52	5A	A9	67	55	17	46	19	F9	AD	38	B5	0E	72
<fauté>	A6	3C	52	DD	E0	0C	55	17	46	57	32	AD	38	B5	C9	49
8ème Tour																
ARK8	F3	6D	48	46	B7	0A	ED	87	98	8E	69	18	76	D2	D7	1E
<fauté>	C5	6D	48	C9	A0	81	ED	87	98	DA	A1	18	76	D2	12	A4
(RK8)	11	14	25	23	66	99	82	4D	44	C8	C5	EB	9C	51	62	79
MC8	E2	79	6D	65	D1	93	6F	CA	DC	46	AC	F3	EA	83	B5	67
<fauté>	D4	79	6D	EA	C6	18	6F	CA	DC	12	64	F3	EA	83	70	DD
SR8	A2	46	10	67	05	91	6E	1D	9F	DC	7E	F8	4B	46	79	CF
<fauté>	26	4A	CB	8D	7D	1D	22	39	2A	F0	20	A3	D8	A3	D1	6E
SB8	A2	46	7E	1D	05	46	79	F8	9F	91	10	CF	4B	DC	6E	67
<fauté>	26	A3	20	39	7D	4A	D1	A3	2A	1D	CB	6E	D8	F0	22	8D
⋮									⋮							
Message	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
<fauté>	1B	8B	BE	9F	7C	BB	87	33	13	C9	33	36	C2	25	2E	B0
Clef	3B	E3	22	66	2F	3B	E8	41	50	2E	79	41	46	05	25	49

FIGURE 3.33 – Extrait de la sortie du programme `reverse_AES.py` appliquée au <fauté> n° 0000 du tableau 3.4 pour chercher à localiser l'opération dans laquelle la faute a été injectée. Les octets devant avoir été modifiés pour produire le <fauté> n° 0000 sont notés en rouge.

L'instant de la perturbation EM a été paramétré pour que celle-ci soit générée durant les opérations qui précèdent l'opération `MC9`. Excepté une désynchronisation du temps d'injection qui aurait pu faire que l'injection se produise après `MC9`, il est fortement probable que l'impulsion EM ait modifié les deux première colonnes du *state* de l'AES à l'issue de la 9ème opération `ShiftRows`.

Nous avons procédé a ce type d'analyse pour les différents `<fauté>` et nous avons essayé de les regrouper en fonction des perturbations les plus probables qui ont pu produire leurs valeurs.

1. Une première catégorie regroupe les valeurs des chiffrés que l'on peut obtenir lorsque la perturbation est introduite après la 9ème opération `ShiftRows`. C'est le cas des `<fauté>` n° 0 et 1 qui peuvent être générés en modifiant les valeurs des octets des deux premières colonnes du `state` de l'AES. Remarquons que seul le chiffré n° 1 est obtenu plusieurs fois dans cette liste et cela pour différents emplacements (x_i, y_j).

2. La deuxième catégorie peut être obtenue lorsque les valeurs des octets sont modifiées après la 9ème opération `MixColumns`. La manière la plus probable d'obtenir le `<fauté>` n° 6 par perturbation est d'effectuer une majorité de *bit reset* sur les deux premières lignes du `state` de l'AES.

3. Une troisième catégorie, plus difficile à expliquer, est composée de chiffrés que l'on obtiendrait si on modifiait un ou deux octets du message initialement utilisé pour nos manipulations. Les chiffrés n° 4 et 7 s'obtiendraient en chiffrant respectivement les messages $M' =$ `0x00000000000000000000000010000000` et $M'' =$ `0x00000000000000000000000003060` avec la clef K. Ces valeurs de chiffrés pourraient également être obtenues si on modifiait un ou deux octets du `state` de l'AES après les premières opérations `AddRoundKey`, `SubBytes` ou `ShiftRows` mais aussi en changeant les valeurs de tous les octets de la dernière colonne de la première opération `MixColumns` ou des deuxièmes opérations `AddRoundKey` ou `SubBytes`. Autant de modifications qui n'expliquent pas le fait que nous injections notre perturbation au niveau du neuvième tour du chiffrement et que l'effet soit antérieur à celui-ci.

4. La dernière catégorie est composée des `<fauté>` que nous n'arrivons pas à expliquer avec les données à notre disposition.

Quoi qu'il en soit, les catégories de fautes 1 et 2 montrent que le réglage temporel de l'injection électromagnétique t_{inj} est satisfaisant puisque la majorité des opérations perturbées sont celles du neuvième tour de l'AES et ce, pour différentes localisations spatiales d'injection. Ces catégories montrent également qu'avec cet injecteur et ces premières expérimentations, nous n'arrivons pas à perturber un unique octet après l'opération `ShiftRows-9`, comme nous nous l'étions imposé.

3.3.3.2.2 Perturbations avec l'injecteur EM Cyl.-∅800-N7

Cet injecteur est illustré sur la figure 3.34. Il a un diamètre approximatif de 0,8 mm autour duquel sont enroulées $N_{Sb} = 7$ spires. Ce modèle a été choisi suite aux expérimentations faites avec l'injecteur EM Cyl.-∅1500-N5. En effet, nous supposons que son diamètre plus fin et ses spires supplémentaires produiront des perturbations plus locales, toujours de manière symétriques autour de l'axe $\vec{e_z}$. La résolution des cartographies a été augmentée afin d'effectuer une analyse plus précise. Le nombre d'essais par point spatial a également été augmenté pour obtenir plus de `<fauté>` et analyser leur répartition spatiale. Ce balayage d'injections électromagnétiques a été effectué avec les paramètres précisés dans le tableau 3.5.

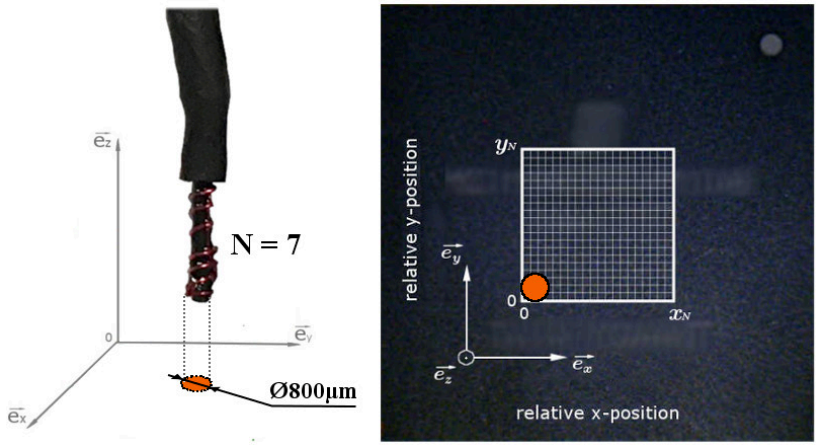


FIGURE 3.34 – Injecteur Cyl.-∅800-N7. La superficie occupé par cet injecteur au-dessus du SoC est la surface orange.

Paramètre	Valeur
Zone balayée : $L = l$	6000 μm
Pas : $\Delta p_x = \Delta p_y$	60 μm
Nombre d'essais par point : N	50
Amplitude de l'impulsion : A_{inj}	$+400V$
Durée de l'impulsion : $\Delta_{T_{inj}}$	$6ns$

} Matrice (101×101) ; 510050 essais

Tableau 3.5 – Paramètres de balayage avec l'injecteur EM Cyl.-∅800-N7

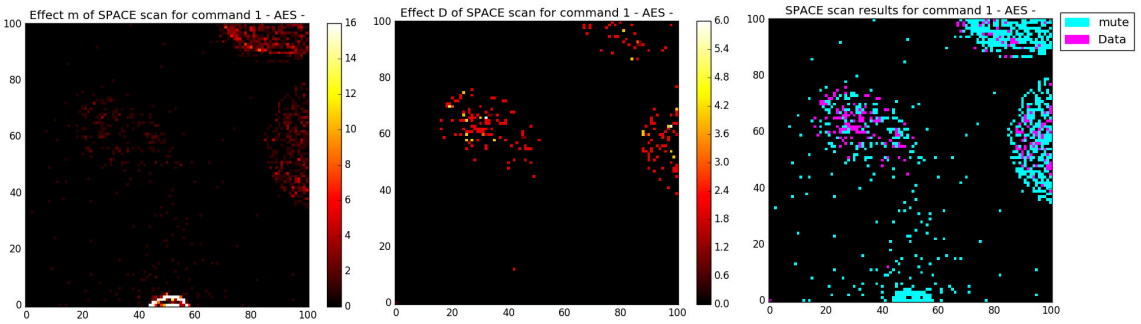


FIGURE 3.35 – Cartographies synthétisant les comportements observés à la surface du SoC lors du balayage avec l'injecteur EM Cyl.-∅800-N7

La surface considérée a été balayée en un peu plus de 19 heures. Sur les 510050 injections électromagnétiques, il y a eu 1917 `<mute>` et 227 `<fauté>`, soit respectivement $0,376\%$ et $0,044\%$ des chiffrements. La figure 3.35 synthétise les résultats obtenus. On distingue 4 zones au-dessus desquelles on obtient des `<mute>`. Deux d'entre elles, situées au niveau du coin supérieur droit, sont sensiblement les mêmes que celles qui ont étés mises en valeur lors de l'expérimentation avec l'injecteur Cyl.- ∅1500-N5, mais avec une sensibilité moindre *i.e.* approximativement 20% des injections électromagnétiques aboutissent à un `<mute>`. La troisième zone, plus centrale, ne présente que quelques « plantages » par rapport au nombre d'injections électromagnétiques ($\approx$ 8% de `<mute>`). La dernière zone représentant deux arcs de cercle concentriques inclus dans le rectangle définit par $\forall(x, y) \in \{[40, 60] \times [0, 10]\}$ est la plus sensible. En effet, pour chaque point de cette zone le nombre de `<mute>` est au maximum de l'échelle du graphique. Cela signifie, qu'en ces points, il y a plus de 16 « plantages » pour 50 injections électromagnétiques. En se référant aux fichiers de `.log` qui ont servi à construire ces graphiques, nous avons observé des points à près de 60% de `<mute>`.

En ce qui concerne les chiffrés `<fauté>`, ils se répartissent selon 3 zones. Deux d'entre elles se superposent avec les zones du coin supérieur droit générant approximativement 20% de `<mute>`. La troisième se superpose avec la zone centrale peu sensible aux `<mute>`. Elles présentent toutes une densité de `<fauté>` à peu près équivalente, avec toutefois celle de la zone centrale qui semble être sensiblement supérieure aux autres.

Analyse des fautes. Les 197 chiffrés erronés se répartissent selon 35 valeurs. Le tableau 3.6 donne uniquement ceux mentionnés dans cette analyse et la liste intégrale est détaillée dans l'Annexe F.2.2. Comme pour l'analyse des fautes obtenues avec l'injecteur EM Cyl.-∅1500-N5, nous avons utilisé le programme `reverse_AES.py` pour essayer de regrouper les `<fauté>` en fonction des perturbations les plus susceptibles de produire ces valeurs. Les deux premières catégories de fautes ont déjà été observées lors de la manipulation précédente.

1. Une première catégorie regroupe les valeurs des chiffrés que l'on peut obtenir lorsque la perturbation est introduite après la $9^{\text{ème}}$ opération `ShiftRows`. Une partie des `<fauté>` peuvent être obtenus lorsque la perturbation électromagnétique modifie une

n° identifant	Valeur du `<fauté>`	(occurences)
(Chiffré Ref.)	`52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13`	
0002	`10 4D A4 00 .. .. .. .. .. .. .. .. .. .. .. ..`	(1 fois)
0007	`.. 4D .. .. .. .. .. .. .. .. .. .. .. .. .. ..`	(4 fois)
0008	`.. 4D .. 8C .. .. .. .. .. .. .. .. .. .. .. ..`	(3 fois)
0011	`.. .. .. .. .. .. .. .. .. .. AD 89 .. .. .. ..`	(1 fois)
0015	`.. .. .. .. .. .. .. .. .. .. AD .. .. .. .. ..`	(3 fois)
0017	`FB 90 FB CD .. .. .. .. .. .. .. .. .. .. .. ..`	(1 fois)
0019	`.. 4D .. 84 .. .. .. .. .. .. .. .. .. .. .. ..`	(2 fois)
0022	`AF 70 BB 63 FD 36 CA 43 F9 85 8A 40 95 DF 5B E9`	(2 fois)
0023	`5A A1 66 26 CD 25 2A 57 07 B8 92 F6 44 20 DE F5`	(22 fois)
0024	`.. 49 .. .. .. .. .. .. .. .. .. .. .. .. .. ..`	(3 fois)
0028	`CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC`	(143 fois)
0032	`07 77 6F CA E6 A8 69 82 CE 37 B6 B1 5E 23 00 6B`	(12 fois)

Tableau 3.6 – Extrait des chiffrés erronés pour l'injecteur EM Cyl.-∅800-N7. Les `".."` indiquent des valeurs d'octets non-modifiées.

ou deux colonnes du `state`. C'est le cas des chiffrés n° 7, 8, 11, 15, 19, 24 dont la plupart ont été relevés plusieurs fois.

2. La seconde catégorie regroupe les chiffrés qui peuvent s'obtenir lorsque les modifications sont introduites après la 9ème opération `MixColumns`. Les `<fauté>` n° 2, 17, 18 et 31 peuvent être obtenus lorsque les valeurs des octets de la diagonale principale du `state` sont modifiées après l'opération `MixColumns-9`.

3. Une nouvelle catégorie de fautes est observée. Elle concerne le transfert du résultat du chiffrement par l'accélérateur matériel à l'adresse mémoire qui lui a été indiquée. Il s'agit des fautes du type n° 28 pour lesquelles l'adresse mémoire destinée à recevoir le chiffré garde la valeur avec laquelle elle avait été initialisée. Il s'agit de l'erreur qui s'est produite le plus grand nombre de fois : 143 fois sur 227 fautes (62,995%).

4. Enfin il y existe une multitude de valeurs que l'on n'arrive pas à expliquer avec les données dont nous disposons. Certaines d'entre elles ont pourtant été reproduites 22 fois pour la n° 23 et 12 fois pour la n° 32.

Cette manipulation a produit des fautes se répétant plusieurs fois et il peut être intéressant d'observer leur localisation spatiale. Elles sont présentées par la figure 3.36. Seul le `<fauté>` n° 28 est commun à toutes les zones au-dessus desquelles les injections électromagnétiques ont généré des perturbations dans la valeur du chiffré. Les fautes n° 23 et 32 que nous n'arrivons pas à interpréter sont pourtant restreintes à des zones bien définies. Des informations plus précises concernant le *layout* de la puce permettrait de comprendre quels IP-bloc sont susceptibles d'être perturbés à ces endroits. Enfin les `<fauté>` liés au 9ème tour du chiffrement AES se répartissent sur un arc de cercle dans le coin droit de la zone balayée. C'est au-dessus de cette zone qu'il faudrait effectuer des cartographies plus

fines afin d'essayer d'appliquer une DFA.

Nous arrivons à générer une multitude de fautes sur le chiffré, cependant l'analyse de celles-ci n'est pas simple et une grande majorité des fautes n'a pas pu être expliquée. Toutefois, avec les multiples occurrences pour certaines valeurs d'entre elles, il a été possible de définir une correspondance entre la valeur de la faute et l'emplacement de l'injecteur électromagnétique lors de la perturbation. Ceci a permis de localiser une zone pour laquelle nous perturbons les opérations du 9ème tour de l'AES. En revanche, ni le nombres de fautes ni leur valeur ne permettent d'appliquer une DFA selon Dussart *et al.*

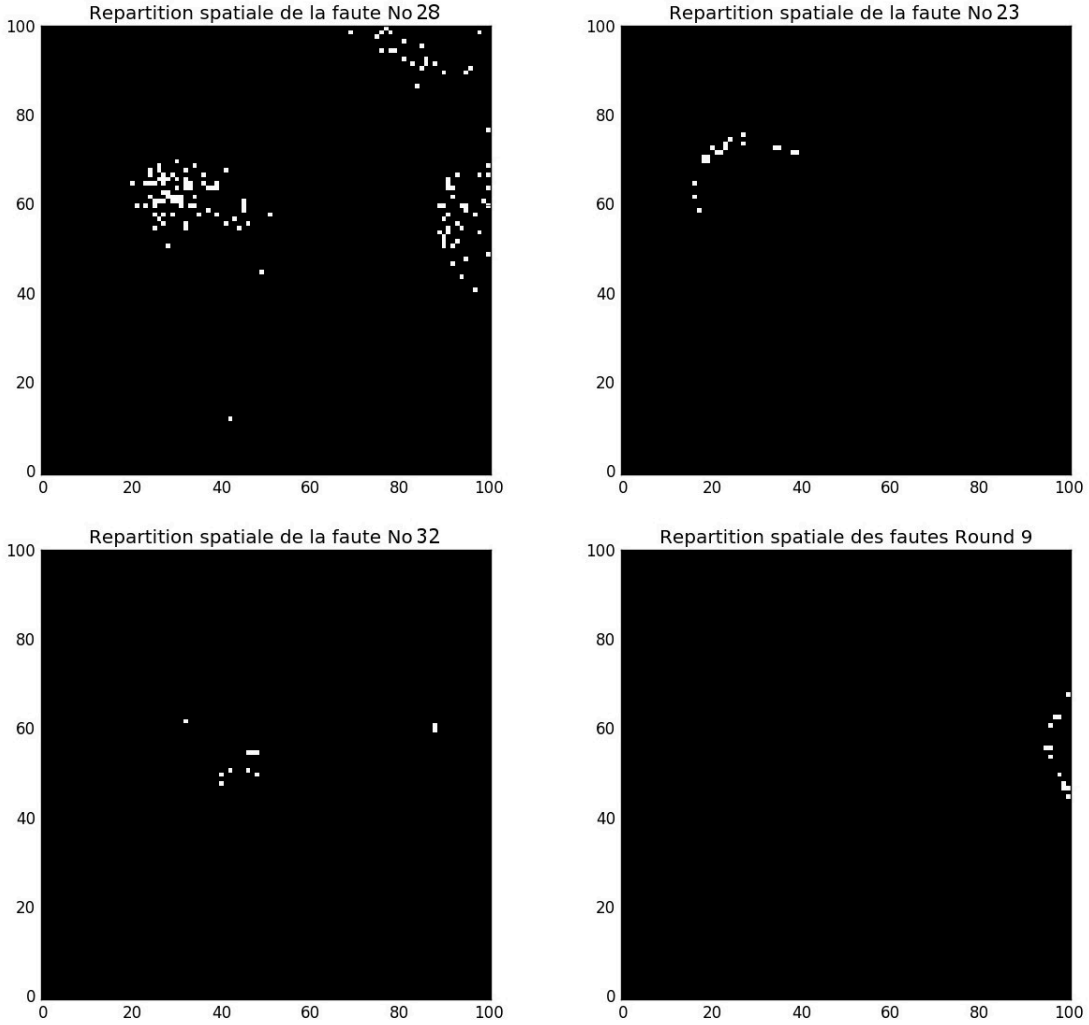


FIGURE 3.36 – Répartition spatiale des chiffrés erronés se produisant plusieurs fois pour l'injecteur EM Cyl.-∅800-N7.

3.3.3.2.3 Perturbations avec l'injecteur EM Ω-∅1800.450-N6

Cet injecteur est illustré figure 3.37. Contrairement aux deux autres modèles décrits précédemment, il possède une composante directionnelle φ autour de l'axe vertical $\vec{e_z}$. En effet, son noyau torique sectionné génère des champs électromagnétiques qui circulent d'une extrémité à l'autre fermant ainsi la portion du tore. On obtient, de cette façon, une perturbation électromagnétique directionnelle, concentrée entre les extrémités de l'injecteur. Le modèle utilisé est constitué de $N_{Sb} = 6$ spires enroulées autour d'un tore sectionné d'approximativement $1,8$ mm de diamètre extérieur avec un écartement entre ses extrémités de $450\mu m$. La résolution et le nombre d'essais pour le balayage sont les mêmes que pour la précédente manipulation car ils ont confirmé un gain d'informations lors de l'analyse des fautes. Pour définir l'orientation la plus appropriée de φ, deux balayages sont effectués avec les paramètres donnés dans le tableau 3.7.

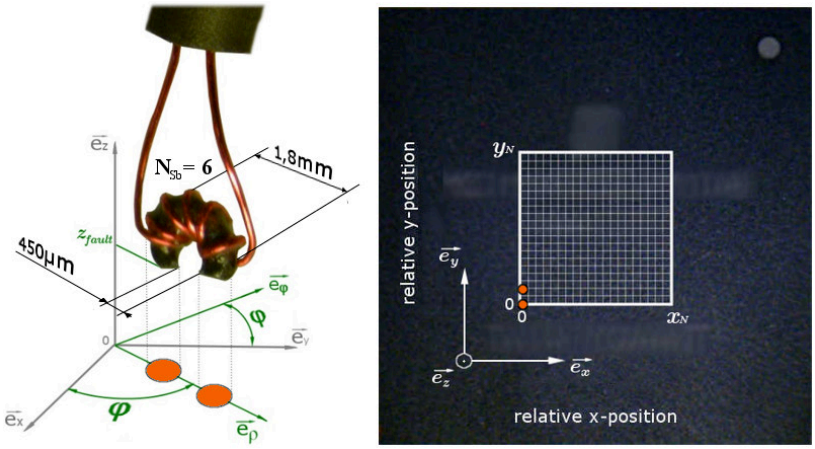


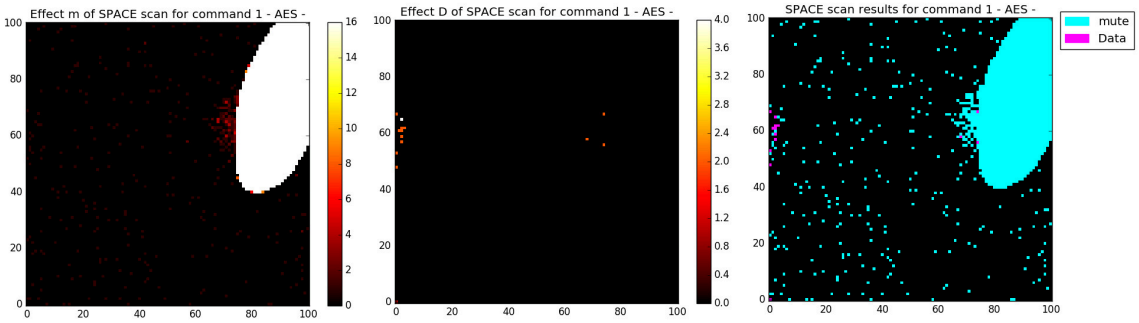
FIGURE 3.37 – Injecteur Ω-∅1800.450-N6. La superficie occupée par cet injecteur au-dessus du SoC est la surface orange.

Paramètre	Valeur	
Zone balayée : $L = l$	6000 μm	} Matrice (101×101) ; 510050 essais
Pas : $\Delta p_x = \Delta p_y$	60 μm	
Nombre d'essais par point : N	50	
Amplitude de l'impulsion : A_{inj}	$+400V$	
Durée de l'impulsion : $\Delta_{T_{inj}}$	$6ns$	

Tableau 3.7 – Paramètres de balayage avec l'injecteur EM Ω-∅1800.450-N6

► **Balayage avec orientation $\varphi = 0$ pour l'injecteur EM Ω-∅1800.450-N6,**

Ce balayage a été effectué en 6 jours, 17 heures et 45 minutes. C'est une durée qui commence à devenir d'autant plus conséquente que nous sommes dans le contexte d'un stade expérimental d'ajustement de paramètres. Sur les 510050 injections électromagnétiques, 64403 ont provoqué un « plantage » complet du système soit 12,627% de `<mute>` et seulement 14 chiffrés comportent des données erronées, soit 0,003% de `<fauté>`. La figure 3.38 résume ces résultats. Si des `<mute>` sont observables sur l'ensemble de la zone balayée, il y en a une au-dessus de laquelle le SoC est extrêmement sensible et plante systématiquement. En effet, dans cette zone, une grande partie des points produisent 50 « plantages » pour 50 perturbations électromagnétiques. La majorité des `<fauté>` est située sur le milieu du bord gauche de la zone balayée.



Nombre de `<mute>` Nombre de `<fauté>` `<mute>`/`<fauté>`

FIGURE 3.38 – Cartographies synthétisant les comportements observés à la surface du SoC lors du balayage avec l'injecteur EM Ω-∅1800.450-N6 ($\varphi = 0$)

n° identifant	Valeur du `<fauté>`	(occurences)
(Chiffré Ref.)	`52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13`	
0000	`CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC`	(5 fois)
0001	`DA C8 C7 BB 8C A7 EC 32 9E 51 30 FE C2 87 97 CC`	(1 fois)
0002	`5A A1 66 26 CD 25 2A 57 07 B8 92 F6 44 20 DE F5`	(8 fois)

Tableau 3.8 – Extrait des chiffrés erronés pour l'injecteur EM Ω-∅1800.450-N6 ($\varphi = 0$). Les `".."` indiquent des valeurs d'octets non modifiées.

Analyse des fautes. Seules 3 valeurs de chiffrés erronés composent les 14 `<fauté>` (voir tableau 3.8). Les valeurs n° 1 et 2 ne parviennent pas à être expliquées et les valeurs n° 0 et 2 ont déja été rencontrées lors de l'expérimentation précédente avec l'injecteur EM Cyl.-∅800-N7 (voir `<fauté>` n° 23 et 28). On souhaite vérifier s'il existe un lien entre les erreurs qui se répètent et leur position (voir Fig. 3.39). Comme pour la manipulation précédente, le `<fauté>` n° 0 est répartit aux différents endroits au-dessus desquels la puce est sensible aux injections. Le `<fauté>` n° 2 est localisé sur une zone restreinte en bordure

de la zone de balayage. Il est intéressant de remarquer que la zone au-dessus de laquelle ce `<fauté>` apparait avec l'injecteur EM Cyl.-∅800-N7 (Fig. 3.36, page 100) n'est pas du tout la même que celle obtenue dans cette configuration. Ceci conforte l'idée qu'il est possible de perturber un même module via différentes zones sensibles aux impulsions électromagnétiques.

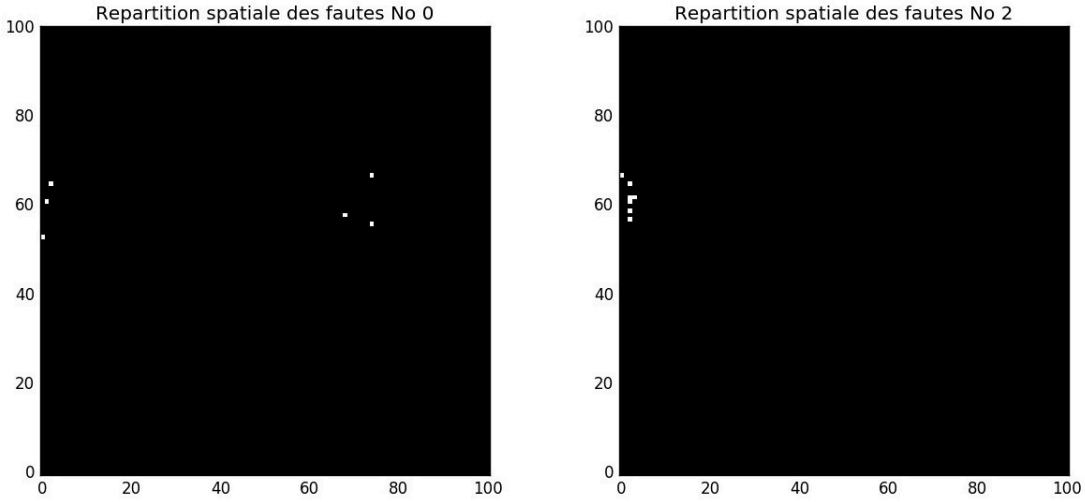
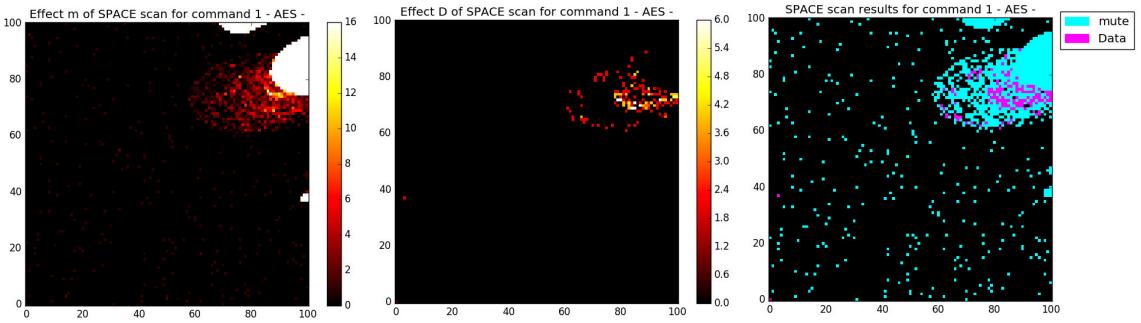


FIGURE 3.39 – Répartition spatiale des chiffrés erronés se produisant plusieurs fois pour l'injecteur EM Ω-∅1800.450-N6 ($\varphi = 0$).

Dans tout les cas, avec cette orientation, l'injecteur Omega n'a pas produit suffisamment de fautes permettant de conclure quant à la possibilité de perturber le chiffrement pour appliquer une DFA. Il faut envisager de renouveler la manipulation en diminuant l'amplitude de l'impulsion A_{inj} afin de vérifier si la zone sensible aux `<mute>` peut générer des fautes exploitables.

► Balayage avec orientation $\varphi = \frac{\pi}{2}$ pour l'injecteur EM Ω-∅1800.450-N6

Le balayage présenté dans ce paragraphe a été effectué avec les mêmes paramètres que la manipulation précédente (voir Tableau 3.7, page 101). Seule l'orientation de l'injecteur change avec $\varphi = \frac{\pi}{2}$. L'ensemble de la cartographie des 510050 injections électromagnétiques a duré 1 jour 15 heures et trente minutes. Sur l'ensemble des injections, il y a eu 14761 `<mute>` et 190 `<fauté>`, soit respectivement $2,894\%$ et $0,037\%$ des chiffrements. La figure 3.40 illustre ces résultats. Des `<mute>` sont observés un peu partout sur l'ensemble de la zone balayée, cependant l'essentiel des zones sensibles pour les « plantages » et la génération d'erreurs dans les chiffrés se situent dans le coin supérieur droit de la cartographie *i.e.* les points sensibles sont contenus dans la zone délimitée par le rectangle défini par $\forall(x, y) \in \{[50, 101] \times [35, 101]\}$.



Nombre de `<mute>` Nombre de `<fauté>` `<mute>`/`<fauté>`

FIGURE 3.40 – Cartographies synthétisant les comportements observés à la surface du SoC lors du balayage avec l'injecteur EM Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$)

Analyse des fautes. On dénombre 38 valeurs de chiffrés erronés pour 190 `<fauté>`. Le tableau 3.9 donne uniquement ceux mentionnés dans cette analyse. La liste intégrale figure dans l'Annexe F.2.3.2.1.

1. La première catégorie de chiffrés fautés concerne ceux qui confirment que notre paramétrage temporel t_{inj} introduit la perturbation durant le 9ème tour de l'AES. Les `<fauté>` n° 4, 6, 29 et 32 sont obtenus lorsque les premières colonnes du `state` de l'AES sont perturbées avant le `MixColumn-9`.

2. On retrouve également la catégorie de `<fauté>` qui indique une erreur de copie du chiffré vers sa destination mémoire. Plusieurs schémas se présentent dans cette manipulation. Le `<fauté>` n° 31 est celui qui a déjà pu être observé dans les manipulations précédentes. Les `<fauté>` n° 16 et 36, montrent que l'impossibilité de copier la valeur du chiffré n'a concerné qu'un ou deux *mots* de 32 bits. Il s'agit bien d'une erreur de copie car le chiffré a correctement été calculé mais mal transféré.

3. La troisième catégorie de fautes souligne les erreurs de transfert du chiffré à proprement parlé. Avec ces erreurs, on observe des portions de la valeur correcte du chiffré attendu mais avec un décalage dans l'indice de positionnement des octets. C'est le cas des `<fauté>` n° 10, 20, 30 et 33. Les chiffrés n° 25 et 34 sont également obtenus suite à des erreurs de transfert mais on observe en plus la modification de valeurs de certains octets.

4. Il persiste toujours des erreurs que nous n'arrivons pas à interpréter. Cependant certaines d'entre elles se produisent plusieurs fois comme les `<fauté>` n° 11 et 12.

Dans cette expérimentation, la majorité des erreurs produites sont des fautes de copie ou de transfert. La faute de copie n° 31 est survenue 51 fois sur 190, soit 26,842% du temps. Celle liée au transfert, la n° 20, a été obtenue 46 fois soit 24,210%. Des fautes ont été produites plusieurs fois à des emplacements différents. Leur localisation spatiale est donnée par la figure 3.41. Cette dernière montre que la n° 31 recouvre l'ensemble de la zone sensible produisant des erreurs dans le chiffré. On observe toutefois des agglomérations de points sensibles par endroits alors que d'autres zones restent insensibles. Les `<fauté>` n° 20, 25

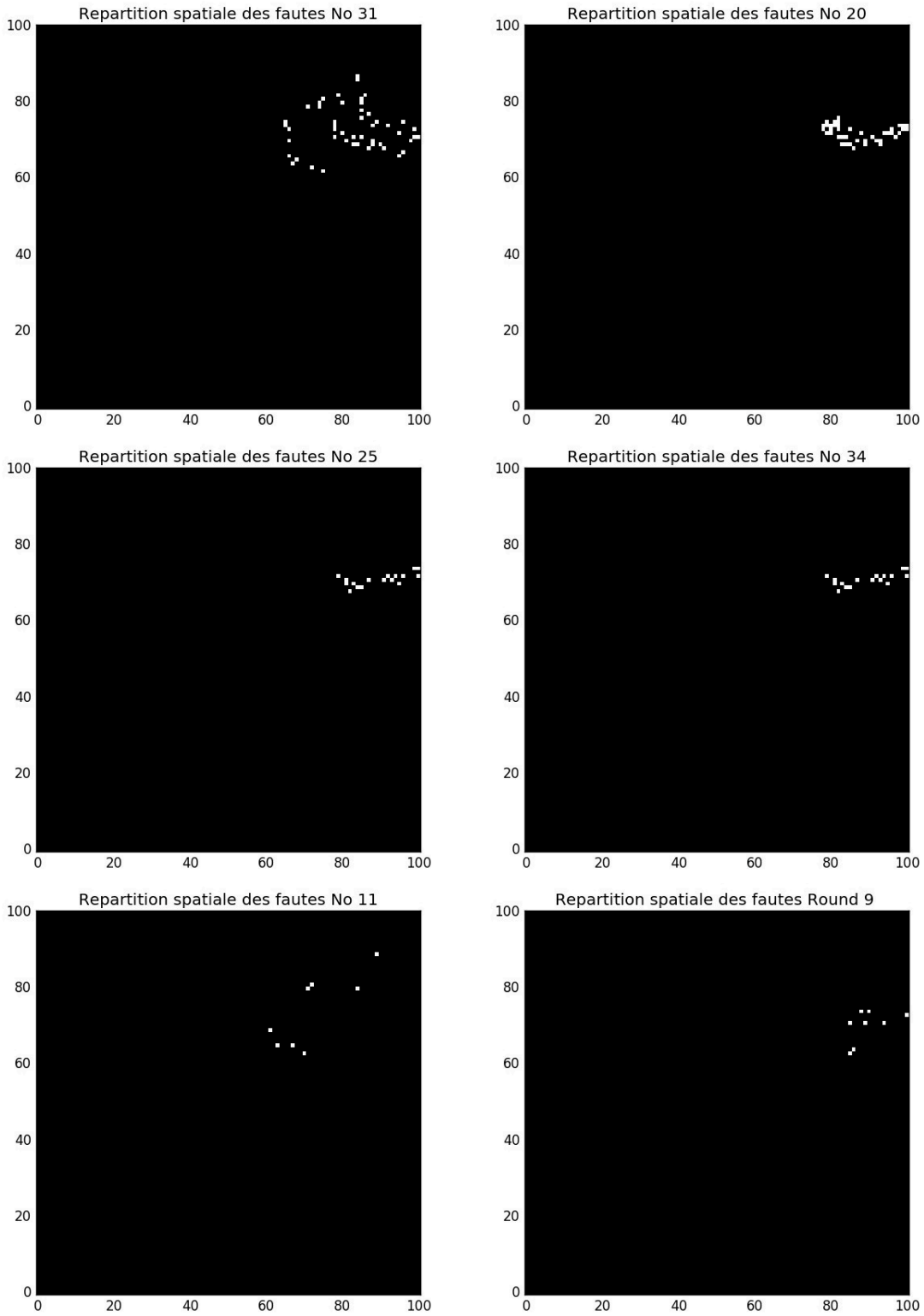


FIGURE 3.41 – Répartition spatiale des chiffrés erronés se produisant plusieurs fois pour l'injecteur EM Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$).

n° identifant	Valeur du <fauté>	(occurences)
(Chiffré Ref.)	52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13	
0004	42	(4 fois)
0006	72 4D	(1 fois)
0010	.. .. 52 4F F4 9C	(2 fois)
0011	38 EB 9C 91 56 F6 08 C9 6D AE E0 F5 E2 8F 02 B6	(8 fois)
0012	72 02 A7 E5 5E CC 6D 7E 04 72 A3 D5 79 A0 0B 7C	(2 fois)
0013	.. .. B1 46 9E 13 00 00 00 00	(2 fois)
0016	CC CC CC CC	(5 fois)
0020	.. .. B1 46 9E 13	(46 fois)
0025	70 01 00 10 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56	(17 fois)
0029	.. .. 45	(1 fois)
0030	.. .. C3 C5 AE 60 B8 A9 81 56	(6 fois)
0031	CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC	(51 fois)
0032	12 .. .. .. 03	(2 fois)
0033	.. .. B8 A9 81 56	(7 fois)
0034	70 01 00 10	(8 fois)
0036	CC CC CC CC 50 CC CC CC	(3 fois)

Tableau 3.9 – Extrait des chiffrés erronés pour l'injecteur EM Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$). Les "." indiquent des valeurs d'octets non modifiées.

et 34 délimitent la même zone confirmant que c'est le même module qui est impacté. De plus cette zone est complémentaire aux zones « insensibles » de la faute n° 31. La n° 11, inexpliquée, ne présente pas de localisation particulière ce qui ne permet pas de conclure quant à son origine. Enfin les fautes liées au neuvième tour de l'AES se regroupent sur un même axe mais n'ont pas de valeur susceptible d'être exploitable dans l'attaque de Dussart *et al.*

3.3.3.2.4 Un besoin de cartographies plus précises

Le balayage de la zone repérant le silicium avec les 3 modèles d'injecteurs présentés précédemment n'ont pas produit le schéma de modifications nécessaire pour l'application de la DFA que nous nous sommes imposée. Cependant, pour chaque manipulation, différentes zones générant des fautes ont été mises en avant. Une multitude de cartographies plus localisées sur ces zones ont été effectuées. A chaque fois, le nombre de modifications du chiffré a été augmenté ainsi que la difficulté pour leur compréhension. Quoi qu'il en soit, le modèle de fautes souhaité a été généré avec l'injecteur Ω-∅1800.450-N6, orienté avec $\varphi = \frac{\pi}{2}$. Ces résultats sont brièvement présentés dans le paragraphe suivant.

► Injecteur EM Ω-∅1800.450-N6, ($\varphi = \frac{\pi}{2}$) - Balayage Zone 1

C'est l'injecteur Ω-∅1800.450-N6 orienté selon $\varphi = \frac{\pi}{2}$ qui a produit les résultats attendus pour cette manipulation. La surface qui a été considérée pour la cartographie d'injec-

tions électromagnétiques est représentée sur la figure 3.42 par le rectangle *Zone 1* en jaune. Avec le système de coordonnées placé à la surface du SoC, l'ensemble des points spatiaux balayés sont les points (x_i, y_j) tels que $\forall (i,j) \in \{[\![58, 101]\!] \times [\![58, 88]\!]\}$. La résolution spatiale a été augmentée afin de produire une matrice de balayage de (101×101) points. Les paramètres utilisés durant cette expérimentation sont résumés dans le tableau 3.10.

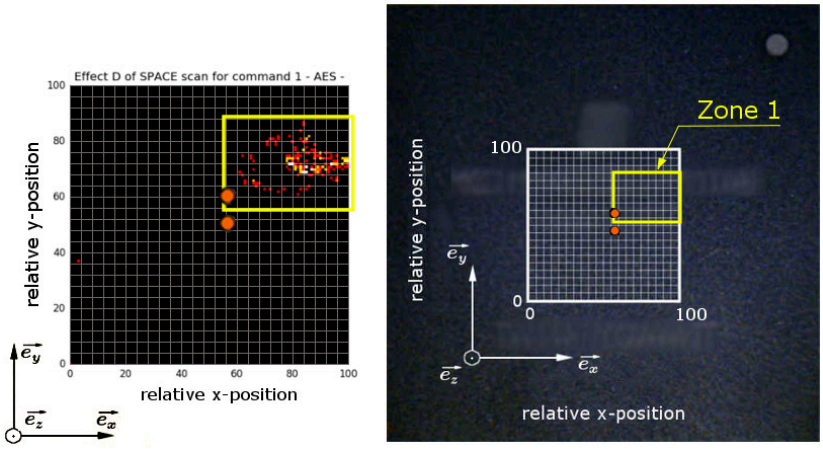


FIGURE 3.42 – La Zone 1 en jaune délimite la surface considérée pour le balayage avec l'injecteur Ω-∅1800.450-N6, $(\varphi = \frac{\pi}{2})$

Paramètre	**Valeur**	
Zone balayée : $L \times l$	$2500 \times 1800\ \mu m$	} Matrice (101×101) ; 510050 essais
Pas : Δp_x ; Δp_y	$25\mu m$; $18\mu m$	
Nombre d'essais par point : N	50	
Amplitude de l'impulsion : A_{inj}	$+400V$	
Durée de l'impulsion : $\Delta_{T_{inj}}$	$6ns$	

Tableau 3.10 – Paramètres de balayage de la Zone 1 avec l'injecteur EM Ω-∅1800.450-N6

La *Zone 1* a été balayée en 4 jours, 1 heure et 20 minutes. Sur les 510050 injections, il y a eu 39305 `<mute>` et 1128 `<fauté>` soit respectivement $7,706\%$ et $0,221\%$ de l'ensemble de chiffrés. La figure 3.43 résume ces résultats. Les mêmes zones sont discernables que lorsque toute la surface de la puce a été balayée. Certains points très sensibles, que ce soit pour les `<mute>` ou les `<fauté>`.

Analyse des fautes. L'intégralité des 103 valeurs de chiffrés erronés sur l'ensemble des 1128 `<fauté>` est donnée dans l'Annexe F.2.3.2.2. Le tableau 3.11 regroupe les chiffrés cités dans cette analyse. Dans l'ensemble, on retrouve les mêmes catégories de `<fauté>` avec des occurrences plus importantes pour chaque chiffré erroné. Ceci est principalement dû à la diminution de la zone de balayage et à l'augmentation de la résolution spatiale.

n° identifant	Valeur du <fauté>	(occurences)
(Chiffré Ref.)	52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13	
0003	CA .. .. 2A .. .. 1D 1A	(1 fois)
0004	DC 49 55 11	(1 fois)
0007	.. .. 11 .. BE ..	(1 fois)
0010	.. .. 52 4F F4 9C	(45 fois)
0016	.. .. 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56	(37 fois)
0020	.. .. AB .. 16	(1 fois)
0026	.. 0D E4	(1 fois)
0027	.. .. BE ..	(4 fois)
0039	93 FB 05 F9 14 0D D7 8B 1C 95 C5 14 AF 52 4F 9B	(8 fois)
0047	.. .. B8 A9 81 56	(338 fois)
0048	.. .. 03	(9 fois)
0049	E4 1D 02 38 AA EA 7B C1 41 36 61 51 47 FD 0F 55	(15 fois)
0057	72 02 A7 E5 5E CC 6D 7E 04 72 A3 D5 79 A0 0B 7C	(6 fois)
0061	E4 23 99 FF 2B 20 97 81 C3 CF 7A 1E 7E 71 D6 3F	(5 fois)
0067	70 01 00 10 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56	(16 fois)
0068	.. .. CC C5 CC CC	(4 fois)
0070	1A EB 84 F4 30 08 FE B0 E0 25 68 D4 7B 6E 39 6F	(8 fois)
0075	CC CC CC CC 50 CC CC CC	(11 fois)
0080	C2 DF 5B D1	(1 fois)
0083	CC CC CC CC 50 00 CC CC	(11 fois)
0084	12 .. .. .. 03	(6 fois)
0085	38 EB 9C 91 56 F6 08 C9 6D AE E0 F5 E2 8F 02 B6	(43 fois)
0086	.. .. AB	(1 fois)
0090	.. .. 31 .. BE ..	(8 fois)
0092	.. .. B1 46 9E 13	(25 fois)
0096	.. .. 45	(1 fois)
0099	CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC	(334 fois)
0101	.. .. 83	(4 fois)
0102	70 01 00 10	(3 fois)

Tableau 3.11 – Extrait des chiffrés erronés pour la Zone 1 avec l'injecteur EM Ω-∅1800.450-N6, ($\varphi = \frac{\pi}{2}$). Les "**..**" indiquent des valeurs d'octets non-modifiées.

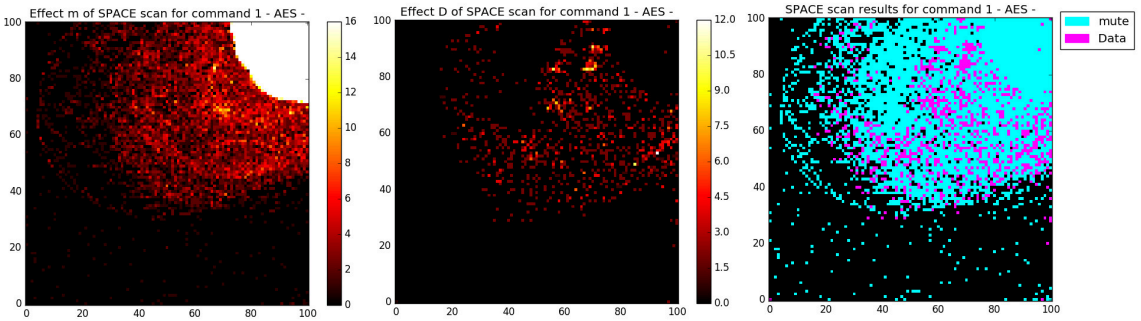


FIGURE 3.43 – Cartographies synthétisant les comportements observés à la surface du SoC lors du balayage de la Zone 1 avec l'injecteur EM Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$)

1. En première catégorie, on retrouve les `<fauté>` causés par une perturbation introduite durant le 9ème tour de l'AES. C'est le cas des fautes n° 3, 7, 20, 26, 27, 48, 84, 86, 90, 96, 101 et 102 qui sont obtenues en introduisant des perturbations sur les colonnes du `state` après l'opération `ShiftRows-9`. La faute n° 3 en particulier a bien le format nécessaire pour l'application de la DFA [50] (le rappel du format des fautes exploitables a été donné dans le Tableau 3.1, page 73). Les modifications n° 4 et 80 sont obtenues en perturbant les valeurs des octets de la diagonale principale après l'opération `MixColumns-9`. Ces deux groupes de `<fauté>` représentent respectivement 40 et 2 valeurs de chiffrés erronés.

2. Les `<fauté>` de cette catégorie correspond aux erreurs de recopie du résultat chiffré. Ces comportements ont déjà été observés. La valeur correcte du chiffré est bien calculée mais la copie de certaines portions est dupliquée. C'est le cas des chiffrés n° 10, 16, 47, 67 et 92. Remarquons que le `<fauté>` n° 47 est celui qui se produit le plus de fois (338 occurences).

3. La catégorie des problèmes de transfert a elle aussi été observée dans les précédentes manipulations. Le coprocesseur n'a pas chargé le chiffré à l'adresse qui lui a été indiquée. Par conséquent, on observe des portions de `"0xCC"` dans la valeur du chiffré. C'est le cas des `<fauté>` n° 68, 75, 83 et 99. Ce dernier représentant le non-chargement de l'intégralité du chiffré est celui qui s'est produit le plus de fois (334 fois).

4. Des comportements que nous ne sommes pas en mesure d'expliquer sont toujours présents. C'est le cas des `<fauté>` n° 39,49,57,61,70 et 85. En particulier, les fautes n° 85 et 57 ont déja été rencontrées dans la cartographie globale de l'injecteur Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$) (respectivement dénotés `<fauté>` n° 11 et 12). Dans cette cartographie, ces deux erreurs se répètent 43 et 6 fois alors qu'elles se renouvelaient 8 et 2 fois dans la cartographie précédente.

La figure 3.44 donne la répartition spatiale des fautes caractéristiques de cette manipulation. A l'instar du balayage de l'intégralité de la surface du silicium, l'erreur de transfert

n° 99 recouvre toute la zone sensible produisant des `<fauté>`. Le problème de recopie du chiffré n° 47 est plus localisé mais chevauche par endroit la zone des erreurs de transfert. La faute inexpliquée n° 85 présente une répartition selon des motifs circulaires concentriques. Enfin, les fautes dues à la modification du `state` de l'AES après l'opération `ShiftRows-9` dessinent deux lignent courbes convergentes en un point.

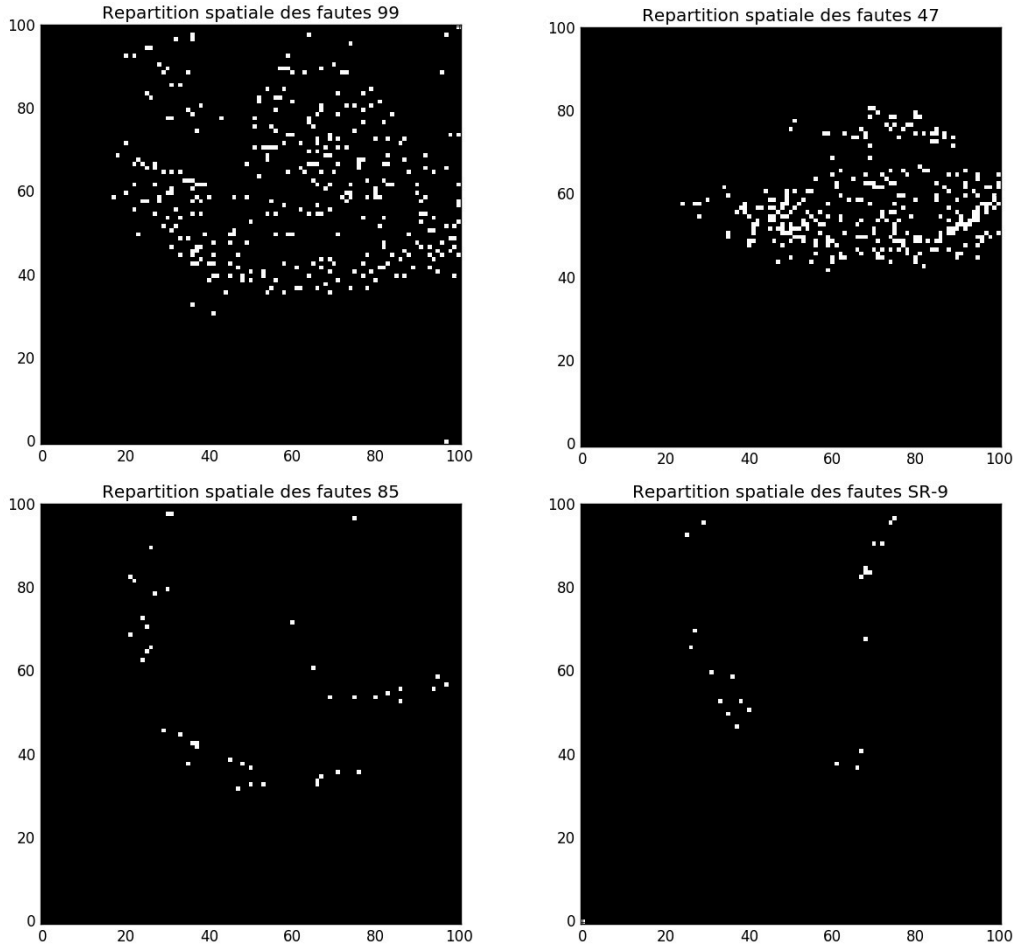


FIGURE 3.44 – Répartition spatiale des chiffrés erronés se produisant plusieurs fois sur la *Zone 1* avec l'injecteur EM Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$).

3.3.3.3 Étude de la faisabilité d'une attaque par injection de faute : conclusion.

Ces multiples cartographies ont montré les difficultés de la mise en place de l'injection de fautes dans le cadre des accélérateurs cryptographiques matériels. Le choix de la sonde n'est pas trivial. Celui des paramètres reste empirique et c'est à force d'expérimentations que nous sommes parvenus à déterminer le modèle d'injecteur susceptible d'effectuer la perturbation souhaitée. D'autre part, l'analyse des fautes générées n'est pas simple. Si nous connaissons les emplacements des modifications d'octets susceptibles d'être exploitables

pour l'attaque sur laquelle nous nous sommes focalisés, il y a une multitude de chiffrés erronés pour lesquels aucune explication rationnelle n'est possible. Cependant, à l'issue de plusieurs cartographies des modifications ciblant le $9^{\text{ème}}$ tour de l'AES ont été identifiées et augmentées. Ceci confirme la faisabilité de l'attaque. En revanche, seule la dernière expérimentation décrite juste avant cette conclusion, révèle un candidat correspondant au modèle de fautes escomptées. La possibilité d'achever une DFA selon Dussart *et al.* semble envisageable mais au prix de nouvelles cartographies ciblées sur les zones sensibles. Ces nouvelles cartographies s'annoncent coûteuses en temps car il faudra ajuster les paramètres de génération d'impulsion électromagnétiques jusqu'alors non automatisé pour arriver au résultat souhaité.

Cette dernière remarque met en évidence la limite de nos manipulations. En effet, la configuration du banc utilisé pour injecter les perturbations ne permet pas d'ajuster dynamiquement l'amplitude $\mathcal{A}_{inj}$ ni la durée des impulsions électromagnétiques $\Delta_{T_{inj}}$. Dans toutes nos expérimentations, pour effectuer les balayages, nous avons été contraints de fixer ces deux paramètres. Cela a eu pour effet de produire certaines cartographies comportant des zones présentant exclusivement des `<mute>`. Or, rien ne permet d'affirmer qu'une zone susceptible de générer des fautes exploitables soit présente à ces emplacements. La bonne pratique voudrait que nous réitérions les différentes opérations de cartographie avec une puissance moindre, jusqu'à ce que des `<fauté>` soient observés ou, du moins, jusqu'à ce que d'avantage de `<mute>` ne soient plus produits systématiquement. Or, cette démarche multiplie le nombre d'injections par autant de valeurs ($\mathcal{A}_{inj}$, $\Delta_{T_{inj}}$) utilisées. Ce n'est pas envisageable lorsque l'on a une contrainte temporelle pour effectuer des tests comme ceux-ci. Pour répondre à ce besoin et effectuer des balayages plus exhaustifs, nous proposons les améliorations suivantes :

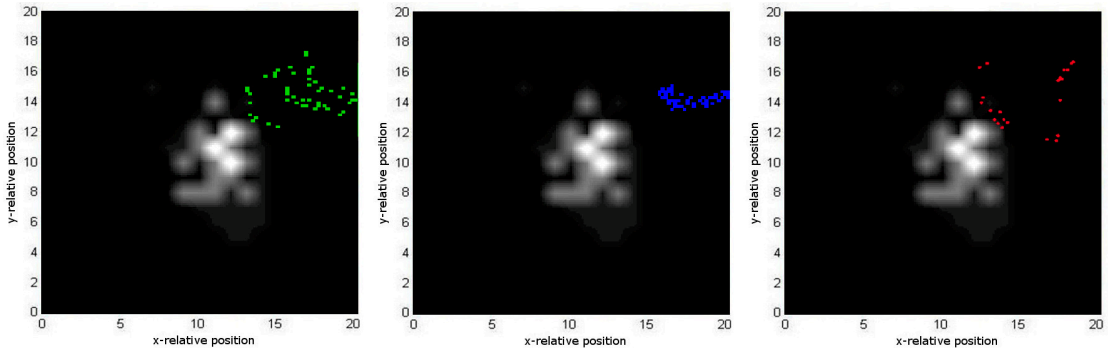
- Dans un premier temps, asservir l'amplitude $\mathcal{A}_{inj}$ au nombre de `<mute>` observés en un point. En fixant des seuils sur la fréquence des « plantages », on peut, par exemple, définir un ajustement de $\mathcal{A}_{inj}$ de manière dichotomique jusqu'à ce que seulement 10% de `<mute>` soient observés.

- Bien que le paramètre $\Delta_{T_{inj}}$ puisse être automatisé, par souci d'optimisation il est plus judicieux de le définir à la suite d'une analyse *side-channel*. En effet, lorsque le processus ciblé a une durée fixe, il est possible de définir un instant pour lequel nous avons la certitude que la perturbation atteindra l'opération désirée sans devoir effectuer de balayage temporel chronophage et pas forcément adapté. En cas d'échec des manipulations, on peut envisager de modifier ce paramètre.

- Généraliser à toute les manipulations une méthode de balayage spatiale plus ciblée permettant de réduire les surfaces balayées. Dans nos diverses expérimentations nous avons effectué une première série de balayages sur toute la surface du SoC avec $\mathcal{A}_{inj} = +400V$. Ces balayages ont permis de différencier les zones sensibles de celles qui ne le sont pas. Un gain de temps est obtenu en ne focalisant les balayages suivants que sur les zones sensibles.

Pour finir, la figure 3.45 montre la comparaison entre les fautes identifiées dans cette section et les zones mises en avant lors de l'analyse *side-channel* section 3.3.1. L'aire au-dessus de

laquelle les émissions du chiffrement AES sont les plus significatives et les emplacements pour lesquels des fautes se produisent ne présente pas une correspondance parfaite. Une partie de cette divergence peut s'expliquer par le fait que les perturbations et les mesures électromagnétiques n'ont pas été effectuées dans les mêmes conditions. Les sondes et les injecteurs possèdent des facteurs de formes différentes qui impactent directement le champ électromagnétique. Une étude plus approfondie pour mieux définir la relation entre les zones d'émissions et les fautes générées permettrait d'améliorer la localisation des zones de génération de fautes exploitables



Faute de transfert Faute de copie mémoire Faute lié au SR-9

FIGURE 3.45 – Comparaison des informations *side-channel* avec les fautes obtenues avec l'injecteur Ω-∅1800.450-N6 ($\varphi = \frac{\pi}{2}$) durant l'évaluation de la faisabilité d'attaquer l'accélérateur cryptographique matériel.

3.4 Conclusion

Dans ce chapitre, nous avons présenté l'application d'attaques matérielles utilisant le canal électromagnétique sur des implémentations cryptographiques.

Notre approche a cherché à évaluer la faisabilité d'attaques matérielles sur un SoC de manière expérimentale selon la méthodologie proposée dans le chapitre 2. Nous avons considéré deux cas pour un chiffrement AES implémenté, tantôt de manière logicielle, tantôt de manière matérielle. L'implémentation logicielle ne comporte pas de contre-mesures et est exécutée par un CPU dont l'architecture et le fonctionnement sont connus. A l'opposé, l'implémentation matérielle est assurée par un accélérateur cryptographique dont la technologie propriétaire indique la présence de protections matérielles contre les attaques par canaux auxiliaires. Une série d'expérimentations *side-channel* et injections de faute ont été appliquées à ces modules.

Pour chacune des expérimentations, l'utilisation de marqueurs temporels a été nécessaire afin de localiser les processus ciblés. Les marqueurs utilisés dans les différentes sections sont des GPIO ou des motifs précis qui permettent de restreindre la quantité de données à traiter pour mettre en place les attaques. Si, dans le cas de l'implémentation logicielle, cette démarche a permis de localiser la signature du calcul AES manuellement, en revanche elle n'a pas été suffisante dans le cas de l'accélérateur matériel. Les signatures électromagnétiques de ce dernier n'ont pu être identifiées qu'à la suite de traitements statistiques automatisés sur toute la surface du SoC. Cette première différence vient du fait que le processeur qui exécute son programme sans contre-mesures est un module qui consomme suffisamment de puissance pour générer une signature électromagnétique capable d'être identifiée visuellement par de simples mesures. A l'inverse, l'accélérateur matériel calcule l'AES de manière optimisée et avec des protections *side-channel* qui rendent l'identification de son travail plus délicate.

En ce qui concerne l'implémentation logicielle sans contre-mesures, l'application d'attaques par canaux auxiliaires semble parfaitement envisageable. Les rayonnements électromagnétiques générés par le CPU sont représentatifs des opérations qu'il exécute avec un RSB suffisant pour procéder à des analyses *side-channel*. Les modèles de fuites classiques qui ont ainsi pu être appliqués ont montré la présence de fuites exploitables dans les divers scénarios d'attaques. En revanche, la détection de fuites sur l'accélérateur cryptographique faite Section 3.3.2 a été plus laborieuse. Il a fallu recourir à divers outils statistiques destinés à la détection de fuites d'informations. L'application de ces outils de manière méthodique a permis d'entrevoir d'éventuelles brèches dans l'implémentation matérielle. Il n'a cependant pas été possible de définir un modèle de fuite adéquat pour pouvoir appliquer une éventuelle attaque dans le temps consacré à cette étude. Ceci illustre bien les difficultés qu'un attaquant peut rencontrer lorsqu'il est confronté à un module sécurisé en « boîte noire ».

La faisabilité d'attaque par injection de faute ne semble pas non plus présenter de difficultés particulières pour l'implémentation logicielle. En utilisant les informations fournies par les analyses *side-channel*, la recherche d'emplacements au-dessus desquels les perturbations injectées produisent le modèle de faute escompté a pu se faire manuellement. Le CPU nécessitant diverses ressources matérielles dans le SoC (horloges, alimentations, etc.)

possède plusieurs bords d'attaques pour perturber ses processus. A contrario, des difficultés ont été rencontrées pour perturber le chiffrement AES de l'accélérateur matériel. Elles ont été détaillées tout au long de la section 3.3.3. Outre l'ajustement des paramètres liés à l'injections EM, nous avons rencontré dans cette partie une difficulté dans le choix des injecteurs. Deux constatations ont été faites à ce niveau. Premièrement en appliquant les suggestions dans le choix des injecteurs de la section 2.3.2.1 du Ch. 2, nous avons observé l'apparition de perturbations se rapprochant de celles que nous souhaitions générer. D'autre part, les même modifications de chiffrés ont été observées avec divers modèles d'injecteurs et à des emplacements différents. Ceci indique que la perturbation d'un module est envisageable de plusieurs manières avec différents modèles d'injecteurs. Cependant une étude plus approfondie pour définir une relation entre les sondes d'observations, les injecteurs EM et les zones sensibles aux perturbations permettrait d'optimiser le choix de l'injecteur.

Les différentes expérimentations appliquées sur l'implémentation logicielle sans contremesures ont montré qu'il était envisageable de mettre en place des attaques matérielles en un temps raisonnable et avec des moyens abordables. En revanche, même si certains des résultats que nous avons obtenus sur l'implémentation matérielle protégée tendent vers une possibilité d'appliquer des attaques matérielles, les expérimentations faites dans ce chapitre ont souligné la nécessité de bénéficier d'un équipement adapté. Entre les deux séries d'expérimentation on observe un ordre de grandeur différent sur la quantité de tests à effectuer pour atteindre l'objectif fixé. Pour optimiser la durée des campagnes de tests sur de tels modules, il est nécessaire d'appliquer une méthodologie adaptée et d'utiliser des bancs automatisés avec une majorité des grandeurs physiques manipulées asservies aux résultats.

Chapitre 4

Attaques matérielles d'environnements d'exécution de confiance

Dans ce chapitre, nous nous intéressons aux mécanismes de sécurité impliqués dans la constitution d'environnements d'exécutions de confiance. Ce type d'environnement est utilisé pour des données ou des processus sensibles devant s'exécuter sur un appareil dont le système d'exploitation ne connaît pas systématiquement l'origine de tous ses programmes. Trois solutions largement répandues dans les SoC d'aujourd'hui sont impliquées dans la mise en place de ces environnements de confiance : le *Secure Boot*, l'extension de sécurité TrustZone® et le *Trusted Execution Environment* (TEE).

1. Le *Secure Boot* est une procédure de démarrage qui vérifie que le système d'exploitation chargé est bien celui qui a été validé par le concepteur du produit final. En l'occurrence, il contrôle l'authenticité du TEE qui va établir la configuration de sécurité de tout le système.

2. TrustZone® est un ensemble de solutions mises à disposition par ARM® dans ses processeurs pour gérer, de manière logicielle et matérielle, l'allocation des niveaux de privilèges aux différents processus.

3. Enfin, le TEE est l'environnement d'exécution de confiance. Il s'agit d'un système d'exploitation minimaliste, isolé, qui supervise la gestion des ressources sensibles entre les différents processus qui les sollicitent. Le TEE s'exécute en parallèle d'un OS plus polyvalent auquel il fournit des services de sécurité au travers d'appels de fonctions. Une partie des TEE s'appuient sur TrustZone® pour renforcer leur isolation.

La Section 4.1 est consacrée au *Secure Boot*, la Section 4.2 à la technologie TrustZone® et la Section 4.3 au *Trusted Execution Environment* utilisant le TrustZone®. Dans chacune de ces sections, nous expliquons le principe de fonctionnement de ces mécanismes ainsi que les vecteurs d'attaques étudiés. Enfin, dans la Section 4.4 nous conclurons ce chapitre.

Sommaire du chapitre

4.1 Secure Boot

Le *Secure Boot* est une procédure de chargement des codes de démarrage d'un processeur qui garantit que le système d'exploitation exécuté sur celui-ci est bien celui autorisé par les concepteurs du système.

4.1.1 Utilisation du Secure Boot

Les SoC utilisent plusieurs types de mémoires dont certaines qui sont externes (pour plus de détails sur le fonctionnement des mémoires dans les SoC, voir : Annexe A.3). Hormis le *Boot ROM code*, qui est le premier code exécuté et qui va charger la séquence de démarrage, les autres chargeurs de codes (*Boot Loader*) ainsi que les systèmes d'exploitation sont stockés dans des mémoires externes. Or, ces composants sont physiquement accessibles et peuvent être détournés comme dans [96] par exemple. En effet, si un attaquant parvient à corrompre la séquence de démarrage pour exécuter son propre code, il aura un accès privilégié aux diverses ressources du système. Il pourra ainsi mettre en place divers scénarios d'attaques et compromettre l'ensemble de la sécurité du système. Par conséquent, dans une démarche de sécurité, les SoC ne peuvent pas faire confiance aux codes qu'ils doivent charger, particulièrement lorsque ceux-ci proviennent de composants externes. C'est pourquoi la méthode de *Secure Boot* est utilisée pour garantir l'authenticité et l'intégrité des divers codes qui vont être exécutés.

4.1.2 Principe de fonctionnement

Nous allons brièvement présenter le fonctionnement du *Secure Boot* afin d'introduire par la suite les chemins d'attaques que nous avons étudiés. Les explications données dans cette section n'entrent pas dans les détails et ne prennent pas en compte toutes les variantes des implémentations existantes. Chaque fabriquant de SoC met en place une séquence de démarrage sécurisée qui lui est propre et il n'existe actuellement pas de normes qui définissent cette procédure. Toutefois, les principes de celles-ci suivent un schéma générique.

4.1.2.1 Plusieurs logiciels pour le démarrage

Les SoC proposés par un fabricant sont destinés à être intégrés dans différents systèmes électroniques élaborés par d'autres entreprises. Par conséquent, ils doivent être configurables afin de gérer les multiples ressources matérielles présentes dans les circuits. En particulier, lors du démarrage, le SoC doit aller chercher le système d'exploitation dans une mémoire non volatile externe et le charger dans sa mémoire principale. Rien que pour ce cas de figure, une multitude de paramètres peuvent changer d'un circuit à l'autre, comme par exemple les types de mémoires, les adresses de chargement, les tailles de codes, etc. Pour permettre une modularité maximum, les concepteurs de SoC proposent un mode de démarrage dans lequel plusieurs *Boot Loaders* élémentaires sont exécutés à la suite. Ces derniers sont des microprogrammes paramétrables qui possèdent deux fonctions principales : initialiser des composants et charger un exécutable qui utilise ces derniers. Le premier *Boot Loader* est le code de la ROM programmé par le fabricant du SoC qui va, par exemple, initialiser les horloges et les mémoires internes. Ce code va ensuite passer la

main à un second *Boot Loader* qui aura été paramétré et chargé au préalable dans une NVM par les concepteurs du circuit intégré. Ces paramètres vont servir, entre autres, à initialiser diverses périphériques et transférer un troisième *Boot Loader* depuis une NVM externe vers la mémoire principale. Ce dernier exécutable prend ensuite la main, termine les diverses initialisations puis charge le système d'exploitation. Ces différents niveaux de démarrage permettent de pouvoir configurer un SoC en fonction des ressources matérielles présentes dans le circuit sur lequel il est implanté.

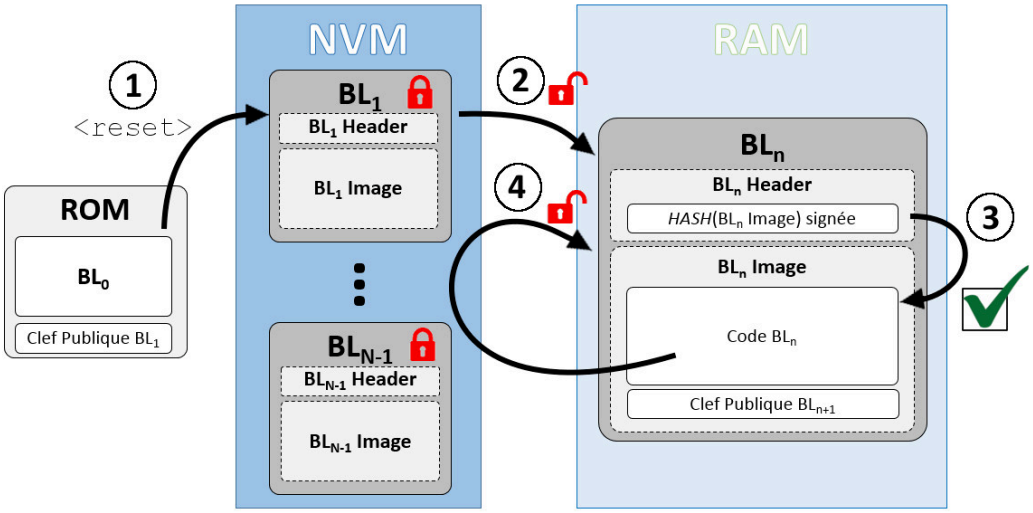


FIGURE 4.1 – Shéma générique d'une séquence de démarrage *Secure Boot*. La ROM est toujours interne au SoC. La première couche de *Boot Loader* s'effectue dans la NVM et la RAM interne. Puis pour les couches suivantes ce sont les NVM et les RAM externes qui sont utilisées.

4.1.2.2 La sécurité sur différents niveaux

La méthode du *Secure Boot* appliquée à cette chaîne de démarrage est une manière de coordonner les divers *Boot Loaders* qui vont être chargés afin de garantir leur intégrité et leur authenticité. Pour cela, avant l'exécution d'un code, celui-ci sera vérifié par le code qui l'aura chargé et ce, jusqu'au démarrage de l'OS. L'étape initiale est supposée fiable puisqu'il s'agit d'un code immutable exécuté à partir de la ROM. La figure 4.1 donne le schéma générique du *Secure Boot*. Sur celle-ci, on a les étapes suivantes :

(1) Le code de la ROM noté BL_0 initialise les paramètres de différents bloc-IP internes au SoC (ex : mémoires internes, activation des horloges). Puis, il va chercher le premier niveau de *Boot Loader* BL_1 dans la FLASH interne et le charge dans la RAM interne du SoC. Par sécurité, la plupart du temps les $BL_{1 \leq n \leq N-1}$ sont chiffrés avec une cryptographie symétrique.

(2) BL_0 déchiffre BL_1 en faisant appel à une routine qui utilise une clé secrète de chiffrement symétrique, souvent stockée de manière sécurisée dans les OTP. Le cas échéant, cette tâche est accomplie par un accélérateur cryptographique.

③ BL_0 vérifie l'authenticité et l'intégrité de l'image de BL_1. Premièrement, BL_0 calcule un haché de l'image de BL_1. Puis avec une clef publique, elle aussi stockée dans les OTP, il vérifie l'authenticité de la valeur du haché fourni dans le *Header* de BL_1. Si la valeur du haché signé après vérification avec la clef publique est égale à la valeur du haché calculé précédemment, alors l'image de BL_1 est intègre, authentique et peut être exécutée. Ce dernier va paramétrer la communication avec divers composants externes comme la NVM et la mémoire principale, puis va procéder au chargement du *Boot Loader* suivant.

④ Jusqu'à ce que l'on démarre l'OS, chaque $BL_{1<n\leq N-1}$ va charger puis déchiffrer BL_{n+1} selon la routine de l'étape ②. Pour que BL_n puisse vérifier l'intégrité et authenticité de BL_{n+1}, la clé publique de BL_{n+1} est intégrée dans son code. En général il y a 2 ou 3 niveaux de *Boot Loader* dans les SoC du marché.

Si à un moment donné, une vérification échoue, alors la séquence de démarrage s'arrête. De cette manière, aucun code non-prévu par le fabricant ne peut être chargé. Cependant, comme pour les algorithmes cryptographiques, l'implémentation du *Secure Boot* est dépendante du matériel sur laquelle il s'exécute. Nous allons maintenant présenter les attaques que nous avons effectuées dans le cadre de cette étude.

4.1.3 Etude de la faisabilité d'attaquer le Secure Boot

Les attaques présentées dans les sections 4.1.3.1 et 4.1.3.2 exploitent respectivement deux types de vulnérabilités. La première exploite une propriété inhérente aux processeurs ARMv7 tandis que la deuxième se base sur une vulnérabilité dans l'implémentation d'un code de mémoire ROM.

4.1.3.1 Attaque exploitant une vulnérabilité de l'architecture ARMv7

Le chemin d'attaque de cette étude a été proposé par Timmers *et al.* dans [140]. Dans ce travail, les auteurs proposent d'exploiter une propriété du jeu d'instructions ARMv7 pour effectuer une attaque combinée qui contourne le *Secure Boot*. Les perturbations de l'attaque sont générées par des *glitches* sur l'alimentation principale du SoC. Pour cela, le circuit a dû être modifié de manière *invasive*, les capacités de découplages ont été dessoudées et le SoC est directement alimenté par une alimentation stabilisée. Dans le contexte de notre étude et avec les contraintes d'expérimentations que nous nous sommes fixées, nous avons étudié la faisabilité d'effectuer cette attaque de manière *non-invasive* en utilisant le canal électromagnétique.

4.1.3.1.1 Principe de l'attaque.

Cette attaque exploite le fait que dans les processeurs ARMv7 il est possible de manipuler la valeur du registre pointeur d'instruction (`pc`) avec certaines opérations autres que les branchements (la description des registres CPU de cette architecture est donnée dans l'Annexe A.2.1). Cela a pour conséquence qu'un registre généraliste, `r15`, est utilisé comme

`pc` ce qui est spécifique à l'architecture ARM. Un code « malicieux » composé d'une première partie exécutable, suivie, de la répétition d'une même valeur est mis à la place d'un des *Boot Loader* de la NVM externe. L'astuce réside dans le fait que la valeur répétée est l'adresse mémoire dans laquelle est stockée la valeur de l'adresse à laquelle le *Boot Loader* remplacé aurait dû être chargé. Le *Secure Boot* va transférer ce code depuis la NVM vers la mémoire principale pour le vérifier. Une partie de ce transfert va être opéré par des instructions de chargement/déchargement des registres.

En effectuant une perturbation durant le transfert, précisément durant la copie des valeurs répétées du pointeur d'adresse, il est possible de modifier l'opérande de l'instruction de chargement pour que celui-ci ne soit plus un des registres généraux (`r0-r12`) mais directement le pointeur d'instruction (`r15`). Ainsi, la valeur chargée dans ce dernier est l'adresse mémoire à laquelle se trouve la partie exécutable du code malicieux. De cette manière, le flot d'exécution du *Secure Boot* est détourné vers le code malicieux qui va pouvoir s'exécuter sans avoir été validé.

4.1.3.1.2 Détails techniques.

Afin de comprendre précisément comment il est possible d'allouer une adresse arbitraire au *program counter* durant le *Secure Boot*, il est nécessaire de détailler certaines instructions ARMv7 [89].

Dans cette architecture, outre les instructions de branchement, quelques instructions spécifiques peuvent également modifier la valeur du registre `pc` : `LDR`, `MOV`, `ADD`, `SUBS`, etc. Deux d'entre elles vont être répétées durant la copie du *Boot Loader*, à savoir les instructions de chargement et de chargements multiples (`LDR` et `LDMIA`). Nous allons expliquer le fonctionnement de l'attaque lorsque l'instruction ciblée est `LDR` mais ceci est valable pour toutes les instructions capables de charger une valeur arbitraire dans le pointeur d'instruction.

LDR<c> <Rt>, [<Rn>{, #+/-<imm12>}]
LDR<c> <Rt>, [<Rn>], #+/-<imm12>
LDR<c> <Rt>, [<Rn>, #+/-<imm12>]!

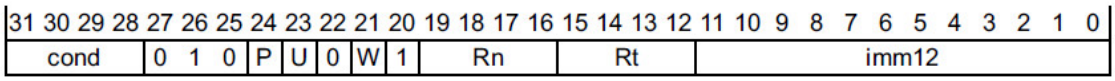


FIGURE 4.2 – Codage de l'instruction assembleur ARMv7 *Load Register*

`LDR` est le mnémonique de l'instruction *Load Register* dont le codage est donné figure 4.2. Cette instruction charge dans le registre `<Rt>` la valeur pointée par l'adresse contenue dans le registre `<Rn>`. Cette adresse peut être pré-indexée ou post-indexée avec un décalage de valeur `<imm12>`. Lorsque la valeur de `<Rt>` est `0b1111`, l'instruction `LDR` charge la valeur pointée par l'adresse contenue dans `<Rn>` directement dans `pc`. Le principe de l'attaque est de perturber une instruction `LDR` pour que `<Rt>` soit forcé à `0b1111` pendant que l'adresse pointée par `<Rn>` est l'adresse du programme que l'attaquant souhaite faire exécuter.

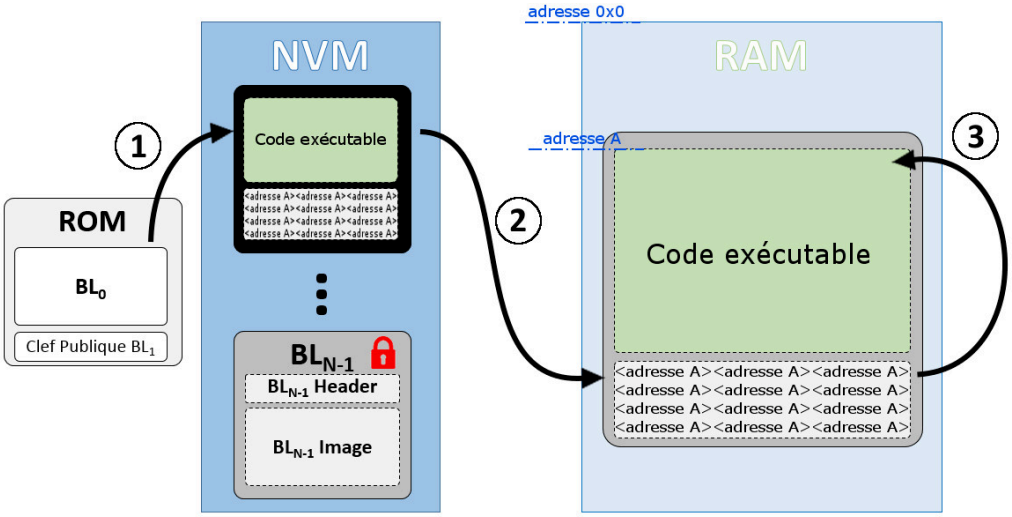


FIGURE 4.3 – Principe de l'attaque exploitant une vulnérabilité de l'architecture ARMv7

La figure 4.3 donne un schéma de principe de l'attaque, et fournit des informations relatives à la structure du code « malicieux ». Ce dernier se termine par la répétition du pointeur `'<adresse A>'`, qui désigne l'adresse contenant la valeur `'adresse A'`, adresse à laquelle le code est chargé par le *Secure Boot*. Cette structure oblige le processeur à charger dans sa mémoire RAM un exécutable, puis à effectuer la répétition de l'instruction `LDR` manipulant la valeur de l'adresse qui pointe vers le début de celui-ci. Cette répétition offre à l'attaquant l'occasion de perturber le processus de transfert afin de détourner le *program counter* et de le faire sauter à l'adresse du début de son code.
En se référant à la numérotation de la figure 4.3, on peut décrire l'attaque par les étapes suivantes :

(1) Sur cette illustration, c'est la séquence de transfert du code BL_1 qui est ciblée. Cependant, cette attaque peut s'appliquer sur les différents *Boot Loader* BL_n impliqués dans le *Secure Boot* tant que BL_{n+1} est substitué par le code « malicieux ».

(2) Le *Boot Loader* BL_n charge le code « malicieux » depuis une mémoire non volatile vers une mémoire volatile à l'adresse de valeur `'adresse A'`. Ce processus est effectué avant toutes les vérifications et le code est copié tel quel. Même si un chiffrement symétrique est utilisé dans la séquence de *Secure Boot*, le déchiffrement n'a pas encore été effectué. C'est précisément pendant l'étape de copie des pointeurs `'<adresse A>'` que la perturbation doit être générée. Le succès de l'attaque dépend de la modification d'au-moins une instruction.

(3) Si une perturbation modifie l'instruction `LDR <Rt>,[Rn]` en `LDR pc,[Rn]` avec $[Rn]_{0 \leq n \leq 12}$ pointant sur la valeur `'adresse A'`, alors la prochaine instruction qui sera exécutée par le CPU est celle qui se trouve à l'`'adresse A'`. C'est précisément le début du code exécutable non vérifié par le *Secure Boot*. En revanche, si aucune perturbation ne parvient à effectuer cette modification, alors les vérifications du *Secure Boot* vont détecter la non-conformité du code et interrompre le démarrage du SoC.

4.1.3.1.3 Evaluation de la faisabilité de cette attaque par perturbations électromagnétiques.

Pour évaluer la faisabilité de l'attaque par injections électromagnétiques, nous allons procéder de la même manière que Timmers *et al.* dans [140]. Nous allons évaluer la possibilité de modifier une instruction `LDR` pour que celle-ci détourne le `pc` à une adresse arbitraire. Le programme de test a été développé pour exécuter une succession d'instructions `LDR`, instrumentées de manière à faciliter les manipulations.

Implémentation du programme de test

Pour effectuer cette évaluation il est nécessaire de manipuler des portions de codes proches du langage machine. D'autre part, pour synchroniser l'injection de la perturbation avec la séquence de `LDR`, il faut restreindre le nombre de processus « parasites » susceptibles de venir perturber l'exécution du programme de test. C'est principalement pour ces raisons que nous avons codé le programme de test en mode *bare metal*. Le pseudo-code 1 résume le fonctionnement de ce dernier.

pseudo-code 1: PROGRAMME DE TEST *bare metal*

Input: Un paramètre reçu par l'UART : $1 \leq n \leq 12$
Output: Une chaîne de caractère de 15 octets : `str_out`.

```
   /* Initialisations */
1  str_PC  ← 0x50432068696A61636B656420212121;      //(15 octets)
2  str_out ← 0xDEADDEADDEADDEADDEADDEADDEADDE;      //(15 octets)
3  GPIO("bas");

   /* main */
4  main(n) :
5  |  GPIO("haut");                 //voir section 2.3.1.2
6  |  Exécuter la routine : LDRrn ;  //voir Fig.4.4
7  |  GPIO("bas");
8  |  str_out ← nnnnnnnnnnnnnnn; //(15 octets de valeur n)
9  return str_out ; // Renvoie str_out sur l'UART du SoC

   /* Fonction non appelée */
10 Fonction print_str_pc :
11 |  str_out ← str_PC;
12 return str_out ; // Renvoie str_out sur l'UART du SoC
```

Avec en particulier :

- Une chaîne de caractères `char[15] str_PC = "PC hijacked!!!"` qui est déclarée au début du programme de test. Après la compilation, cette chaîne sera placée dans la partie `.data` du programme, précisément à l'adresse `0x1020_0000`.

- Une fonction assembleur `print_str_pc` codée mais qui n'est jamais appelée. Cette fonction renvoie sur le port série du SoC la chaîne de caractères `str_PC`. Après la compilation du code, son adresse dans le programme de test est `0x100F_741C`.

- Un `main()` qui prend en paramètre une valeur $1 \leq n \leq 12$ et exécute la routine assembleur `LDRr`n correspondante (voir Fig. 4.4). Cette routine va en particulier charger l'adresse qui contient la valeur de l'adresse de la fonction `print_str_pc` dans `r0` puis, effectuer 1000 répétions de l'instruction `LDR r`n`, [r0]`.

```
        .func LDRrn
LDRrn:
        MOVW r0, #0x1020
        MOVT r0, #0x5000
        NOP
        ...
        NOP
        LDR rn, [r0]
        ...
        LDR rn, [r0]
        NOP
        ...
        NOP
        MOV PC, LR
        .endfunc
```

FIGURE 4.4 – Routine assembleur des différentes fonctions `LDRr`n`, avec` $1 \leq n \leq 12$. Le registre `r0` est chargé avec la valeur `0x10205000`, qui est l'adresse contenant la valeur de l'adresse de la fonction `print_str_pc` (`0x100F741C`)

Dans le cadre du fonctionnement normal, une valeur n est transmise au programme à travers l'UART, puis le `main()` retourne une chaîne de 15 octets répétant cette valeur. Si une perturbation modifie "`LDR r`n`, [r0]`" en "`LDR pc, [r0]`", puisque `r0` pointe vers l'adresse de la fonction qui renvoie `str_PC`, alors la chaîne de 15 octets retournée sera `"0x50432068696A61636B656420212121"`. Ceci permet de détecter que la modification `r`n en `r15` (`pc`) a bien eu lieu.

Comme pour les différentes manipulations effectuées durant cette étude, le programme est prévu pour fonctionner dans l'environnement de test des bancs EM décrit dans le chapitre 2. Les données entre la carte de développement et le Raspberry Pi sont échangées à travers l'UART. Comme nous l'avons expliqué dans la Section 2.3.1.2 du Ch.2, pour

localiser temporellement l'exécution de la routine `LDRr`n, nous utilisons un signal GPIO et des instructions `NOP`.

Véhicule de test

Le véhicule de test utilisé dans cette expérimentation est le même que celui du Ch.3. En effet, le processeur principal intégré dans ce SoC est un processeur d'architecture ARMv7. De cette manière, les informations nécessaires au développement du programme *bare metal* sont en partie déjà disponibles.

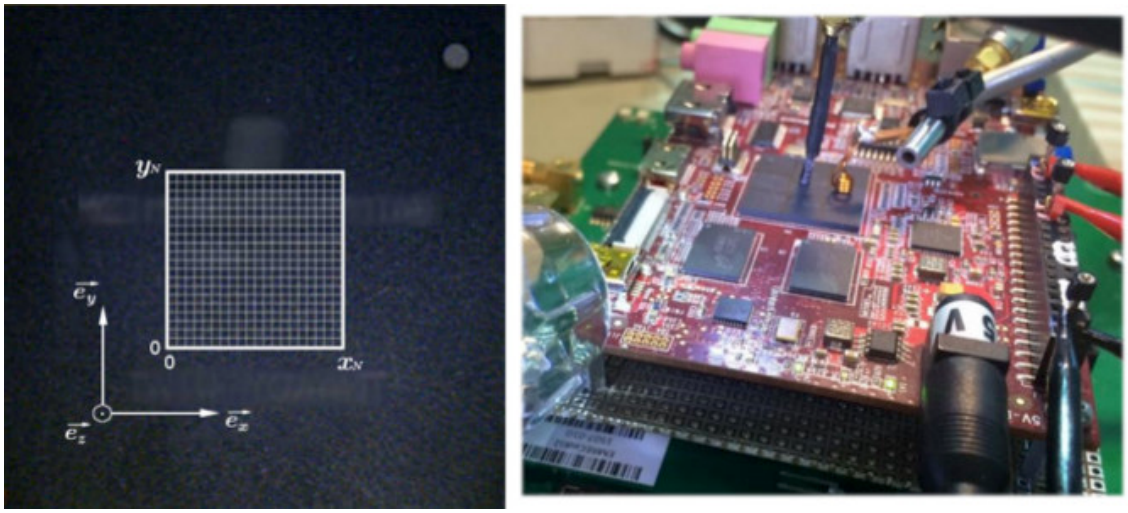


FIGURE 4.5 – Surface du SoC considérée pour le balayage d'injection EM

Paramètre	**Valeur**
Zone balayée : $L = l$	6000 μm
Pas : $\Delta p_x = \Delta p_y$	60 μm
Nombre d'essais par point : N	10
Amplitude de l'impulsion : A_{inj}	$+360V$
Durée de l'impulsion : $\Delta_{T_{inj}}$	$6ns$

} Matrice (101×101) ; 102010 essais

Tableau 4.1 – Paramètres de balayage avec l'injecteur EM Cyl.-∅850-N9

Procédure d'injection de faute

La procédure d'injection électromagnétique a été effectuée avec le banc d'injection EM (voir Ch.2, Section 2.5). Cette procédure consiste à exécuter un script qui envoie successivement les 12 valeurs de n et attend 12 réponses du véhicule de test. Durant chaque envoi d'une des valeurs n, une impulsion électromagnétique d'amplitude $A_{inj} = +360V$ et d'une durée ΔT_{inj} est générée. Cette opération est effectuée 10 fois au-dessus d'un point

spatial du SoC. L'ensemble de la puce a été testé point à point. Cela représente une surface carrée de 101×101 points espacés de $60\mu m$ les uns des autres. Le quadrillage blanc de la figure 4.5 illustre ces points et le tableau 4.1 synthétise les paramètres de la manipulation.

Une procédure d'analyse *side-channel* préalable a permis de définir le temps d'injection t_{inj}. En effet, les mesures ont rapidement montré des motifs caractéristiques entre la montée et la descente du GPIO, permettant ainsi de localiser la répétition des 1000 `LDR`. L'instant d'injection a été défini de manière à cibler le centre de cette répétition. De cette façon, même si des interruptions devaient décaler l'exécution des `LDR`, nous sommes certains qu'un maximum d'injections électromagnétiques se produiront pendant les `LDR`.

Perturbations avec l'injecteur EM Cyl.-∅850-N9

L'ensemble des injecteurs dont nous disposons pour cette étude est décrit au point (2) de la Section 2.5 du Ch.2. Les résultats présentés par la suite ont été obtenus avec l'injecteur électromagnétique cylindrique, d'un diamètre approximatif de $850\mu m$, autour duquel 9 spires d'un fil de cuivre sont enroulées.

► **Résultats expérimentaux**

Les $36.10^6 \mu m^2$ de la surface considérée ont été balayés en 45 heures et 32 minutes. Sur les 1224120 perturbations électromagnétiques, il y a eu 3868 `<mute>` soit 0.316% et le registre `pc` a été détourné 33 fois. La figure 4.6 présente la localisation spatiale des endroits où les registres `r`n ont été modifiés en `pc`. Le nombre de modifications en fonction de la valeur de l'indice du registre utilisé est donné Fig.4.7.

L'observation de ces figures montre l'existence de zones sensibles. La majorité des modifications a été obtenue dans l'aire délimitée par le rectangle $\forall(x, y) \in \{[5, 65] \times [0, 25]\}$. Cependant, il ne semble pas y avoir de relation entre la valeur de l'indice du registre modifié et la localisation spatiale de l'injection. Si on analyse le nombre de fois où `pc` a été modifié en fonction de la valeur de l'indice du registre, on observe que certaines valeurs ressortent par rapport à d'autres. C'est le cas des registres `r11` et `r12` qui ont respectivement été remplacés par `pc` 6 et 8 fois alors que les registres `r4` et `r5` n'ont pas été modifiés. Nous n'avons pas suffisamment d'échantillons pour appliquer des statistiques, mais on remarque une relation avec la distance de Hamming des valeurs d'indices `r`n et `r15` (`pc`) :

n	1	2	3	4	5	6	7	8	9	10	11	12
$H_D(15 \oplus n)$	3	3	2	3	2	2	1	3	2	2	1	2
Nb de `<fauté>` obtenus	1	1	4	0	0	1	5	2	2	3	6	8

Les indices de `r7`, `r11` et `r12`, pour lesquels la modification de l'instruction `LDR` s'est produite à plusieurs reprises, ont une distance de Hamming inférieure ou égale à deux. Il y a moins de bits à mettre à '1' pour passer de `r`n à `r15`. En revanche, nous n'expliquons pas pourquoi `r5` et `r12`, ayant tous deux leur indice d'une distance de Hamming de 2 avec 15, ont des comportement totalement opposés. L'un n'est pas perturbé tandis que l'autre l'est au maximum. Des expérimentations plus approfondies permettraient d'éclaircir ce point mais cela n'est pas nécessaire dans le cadre de cette étude.

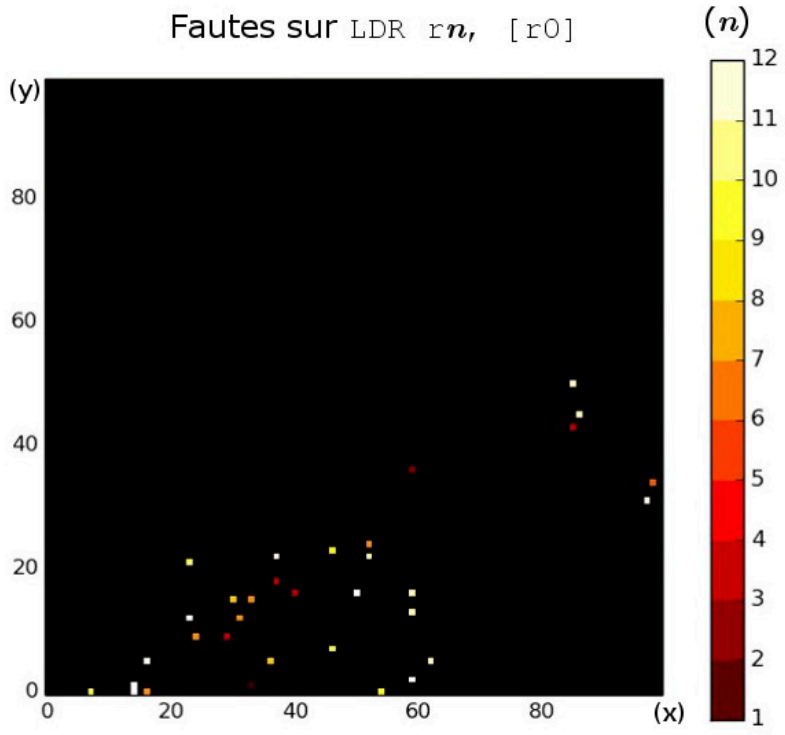


FIGURE 4.6 – Localisation des différentes valeurs de registres `rn` qui ont été modifiées en `pc` dans l'instruction `LDR rn, [r0]`

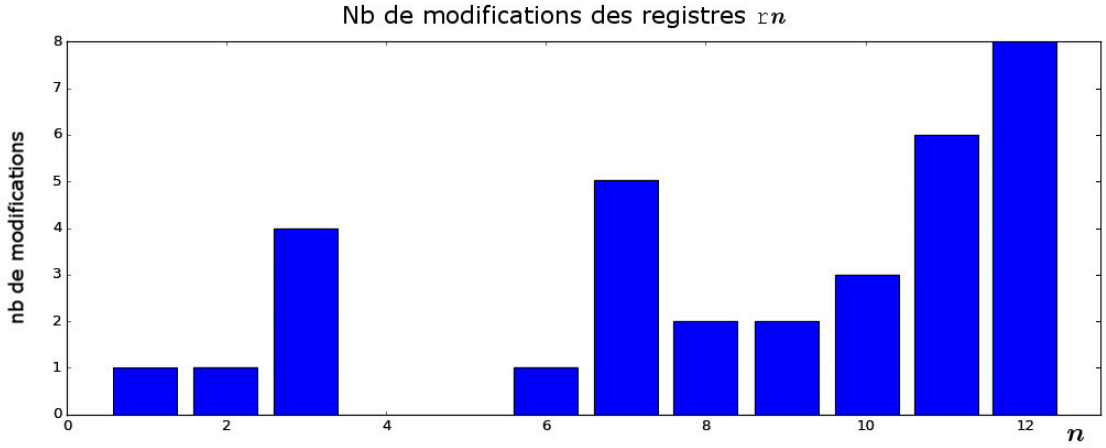


FIGURE 4.7 – Nombre des modifications de l'opérande de `LDR` en `pc` en fonction du registre `rn` utilisé

4.1.3.1.4 Attaque exploitant une vulnérabilité de ARMv7 : Conclusion

En utilisant un code de test instrumenté et des perturbations électromagnétiques, nous avons montré qu'il était possible de modifier l'opération "`LDR rn, [r0]`" en "`LDR pc, [r0]`" et ce, pour différentes valeurs de n. Il semblerait que ce chemin d'attaque pour le *Secure Boot* soit pertinent. Cependant, comme Timmers *et al.* dans [140], nous avons validé notre chemin d'attaque en l'appliquant sur une opération utilisée durant le *Secure Boot*. Nous nous sommes placés dans une configuration expérimentale simplifiée. Reproduire cette manipulation sur un *Secure Boot* réel serait plus complexe. En effet, l'hypothèse sur laquelle se fonde l'attaque est que le transfert des *Boot Loader* soit effectué par des opérations CPU. Or, des systèmes tels que les SoC intègrent des DMA qui déchargent le processeur de ce genre d'opérations, ce dernier n'étant là, la plupart du temps, que pour terminer le transfert des blocs mémoires non alignés. Ceci diminue la durée des répétitions de la copie de l'adresse ajoutée à la fin du code « malicieux ». D'autre part, dans notre contexte expérimental, nous avons implémenté des repères temporels qui nous aident pour synchroniser les impulsions électromagnétiques avec le code exécuté. Dans le cas d'un *Secure Boot* réel, il n'est pas possible de placer de tels marqueurs et une analyse *side-channel* est nécessaire pour localiser les instants des opérations à perturber. L'exploitation de ce chemin d'attaque semble réalisable mais sa mise en œuvre demande une préparation assez conséquente, pouvant durer plusieurs semaines, voire des mois.

4.1.3.2 Attaque exploitant une vulnérabilité d'implémentation du code de la ROM

La vulnérabilité exploitée dans cette section est indépendante de l'architecture du processeur. Il s'agit d'une faiblesse liée aux processus de vérification implémentés dans un code ROM. Nous proposons d'étudier l'élaboration d'une attaque combinée afin de montrer comment il est possible de détecter et d'exploiter de telles vulnérabilités. L'attaque utilise les impulsions électromagnétiques comme vecteur de perturbation et un code « malicieux » basé sur du code propriétaire détourné.

4.1.3.2.1 Véhicule de test - Contexte

L'étude effectuée dans cette section s'est déroulée dans le cadre de l'évaluation sécuritaire d'un ASIC propriétaire. Il s'agit d'un SoC non destiné à une commercialisation grand public. Pour des raisons de confidentialité, nous ne donnerons pas de détails précis liés à ce système. Seuls les principes des méthodes utilisées pour la mise en place de l'attaque seront abordés. Notons que ce système est multitâche, qu'il intègre un processeur ARM Cortex-A7® de 32 bits et qu'il démarre selon une séquence de *Secure Boot*.

4.1.3.2.2 Implémentation des vérifications

La séquence de vérification implémentée sur ce système suit le schéma générique des *Secure Boot* donné dans Section 4.1.2.2. Une nuance d'implémentation décrite dans la figure 4.8 est à noter par rapport à la séquence générique.

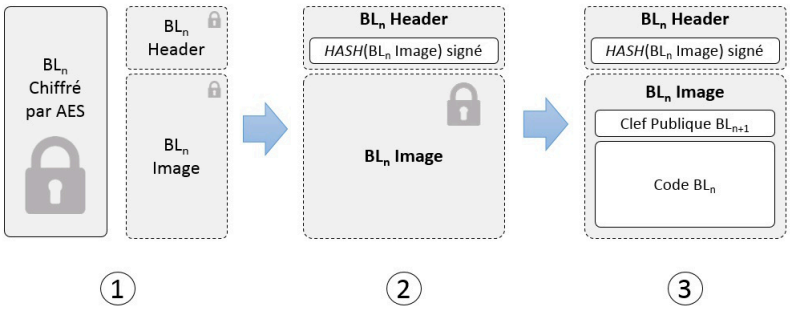


FIGURE 4.8 – Schéma de la séquence de vérifications implémentées

(1) Les différents *Boot Loader* BL_n provenant de l'extérieur du SoC sont protégés par un chiffrement AES. BL_n va charger BL_{n+1} dans la mémoire principale, chiffré. La clef secrète utilisée pour le déchiffrement est stockée de manière sécurisée dans une mémoire OTP accessible uniquement à l'accélérateur matériel du SoC.

(2) La nuance avec la séquence de *Secure Boot* générique est, qu'ici, le déchiffrement du *Boot Loader* se fait en deux temps. D'abord, seul l'en-tête (*Header*) va être déchiffré. Une première série de vérifications est effectuée sur celui-ci. Ces vérifications sont principalement réalisées sur la taille, le format, la version et l'intégrité. Cette dernière est effectuée à l'aide d'un calcul CRC dont la valeur de référence est contenue dans le *Header*. Si l'une de ces vérifications échoue, alors la séquence de *Secure Boot* arrête le démarrage. Sinon, elle passe à la vérification de l'image.

(3) L'image du *Boot Loader* est à son tour déchiffrée et la série de vérifications continue (calcul du haché et comparaison de celui-ci avec le haché de référence signé).

Le fait que le déchiffrage soit effectué en deux étapes met en avant un éventuel point sensible contenant un test conditionnel : la transition de l'étape (2) → (3).

4.1.3.2.3 Elaboration de l'attaque.

Afin de localiser d'éventuelles vulnérabilités dans ce *Secure Boot*, des étapes préalables de rétroconception et d'analyse *side-channel* ont étés effectuées. C'est durant ces étapes que l'attaque, telle qu'elle est présentée dans cette section, a été élaborée.

Parmi les diverses expérimentations, tel des développeurs tierces, nous devions charger les *Boot Loader* dans la mémoire NVM de la plateforme d'évaluation. Nous avions à notre disposition deux jeux de codes binaires, deux plateformes de développement et de

la documentation. Le jeu de codes chiffrés est destiné à la plateforme dont le *Secure Boot* est actif tandis que le jeu non chiffré est destiné à l'autre plateforme. Le mode de sécurité des plateformes est configuré par des valeurs stockées en OTP. Soulignons que, pour des développeurs tierces, il est normal d'avoir à disposition ces codes et ce matériel afin de leur permettre de configurer leurs produits comme ils le souhaitent.

4.1.3.2.4 Analyse des codes binaires

En effectuant une analyse approfondie des binaires des *Boot Loader*, nous avons obtenu les informations suivantes :

- Qu'ils soient sécurisés ou pas, les binaires ont la même taille.
- En comparant les différents codes non chiffrés, nous avons constaté que les *Boot Loaders* sont composés d'un *Header* de 256 octets, suivi de l'image du code.
- Les *Header* non sécurisés ont, au début de leur code, la même valeur notée sur deux octets `<0xBAAB>`.
- Certains emplacements sont remplis avec des '`0x00`', en particulier un qui en contient 96. Il s'agit probablement des espaces réservés aux différentes valeurs nécessaires pour la vérification, comme la valeur du haché signé de l'image par exemple.

Suite à ces constatations, afin d'observer les différents comportements de la séquence de *Secure Boot* en fonction du code vérifié, nous avons chargé diverses configurations de ces codes dans la mémoire FLASH. En effet, en connaissant la taille du *Header*, nous pouvons scinder les codes et les accoler à d'autres.

4.1.3.2.5 Rétroconception side-channel

Les mesures des émissions électromagnétiques ont été effectuées avec le banc de mesures EM (voir Section 2.4 du Ch. 2). La sonde utilisée est une Langer ICR HH-150 de bande passante 6GHz. Le Cortex-A7 de ce SoC émet suffisamment de rayonnements électromagnétiques caractéristiques pour repérer manuellement les zones d'intérêt. Trois *Boot Loaders* ont été utilisés pour les expérimentations *side-channel* :

- `BL_sec:` le *Boot Loader* sécurisé, chiffré, adapté à la plateforme sécurisée exécutant le *Secure Boot*.

- `BL_nonsec :` le *Boot Loader* non sécurisé adapté à la plateforme n'exécutant pas le *Secure Boot*.

- `BL_Hd.sec_IMG.nonsec :` un code combinant les deux. Il est constitué du *Header* du *Boot Loader* sécurisé, et de l'image du *Boot Loader* non sécurisé.

Avec cet ensemble de codes, nous avons cherché à localiser temporellement l'exécution des différents processus impliqués dans le *Secure Boot*. La première manipulation consiste à comparer l'exécution du processus de démarrage sécurisé avec celle du processus non sécurisé. Les mesures de ces deux séquences sont données sur la figure 4.9.

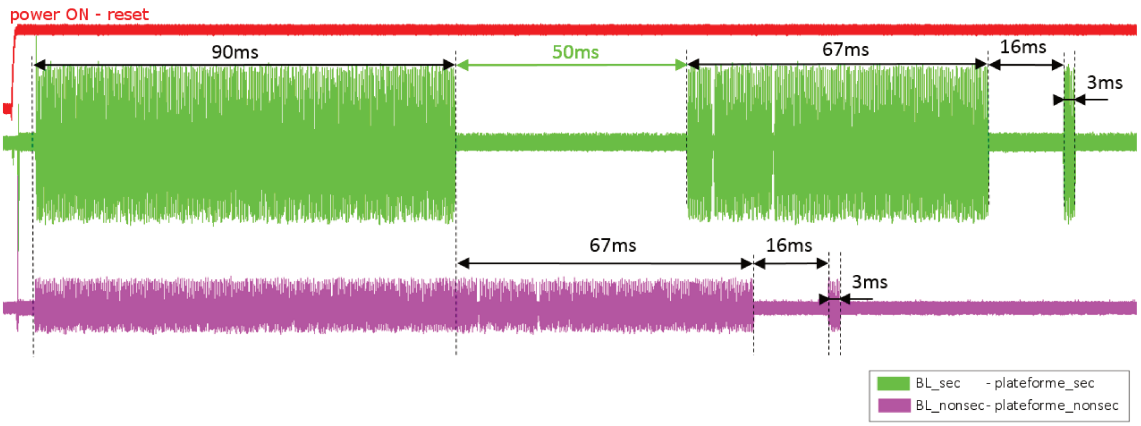


FIGURE 4.9 – Comparaison des mesures EM d'une séquence de démarrage non sécurisée sur la plateforme non sécurisée avec une séquence de *Secure Boot* sur la plateforme sécurisée. (Le calibrage des mesures de l'oscilloscope n'est pas le même pour les deux mesures mais les signaux ont la même amplitude.)

Cette figure fournit plusieurs renseignements. Premièrement, on constate que la séquence sécurisée est plus longue que celle qui ne l'est pas. De manière évidente, la présence des diverses opérations de vérification semblent allonger sa durée. Deuxièmement, en observant les motifs des deux signaux, on remarque que ceux-ci sont similaires point par point hormis un décalage causé par un « creux » de $50ms$ présent dans la séquence sécurisée. Il semblerait, qu'à l'issue des premières $90ms$, une nouvelle phase soit déclenchée et que cette dernière n'émette pas d'informations à l'emplacement où sont effectuées ces mesures. De nouvelles expérimentations vont aider à analyser l'origine de cette divergence de comportement.

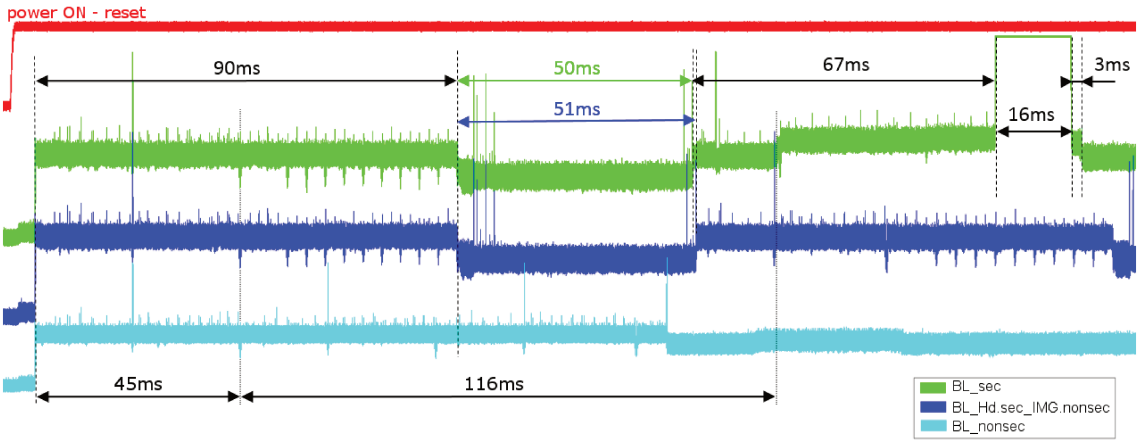


FIGURE 4.10 – Comparaisons de mesures EM filtrées de différentes configurations de *Boot Loader* chargés sur la plateforme sécurisée

Les manipulations sont uniquement effectuées sur la plateforme sécurisée. Seuls les *Boot Loaders* formatés selon les critères des concepteurs de la plateforme sont autorisés à être exécutés sur celle-ci. Nous procédons à trois séquences de démarrages successives en chargeant tour à tour `BL_sec`, `BL_Hd.sec_IMG.nonsec` et `BL_nonsec`. Les mesures EM des trois démarrages sont données sur la figure 4.10.

On observe plusieurs comportements sur celle-ci :

- Les 3 signaux sont identiques durant les premières $45ms$. Après cette limite, seuls `BL_sec` et `BL_Hd.sec_IMG.nonsec` sont similaires.

- Jusqu'à la fin des premières $90ms$, `BL_sec` et `BL_Hd.sec_IMG.nonsec` ont le même profil de signal puis une différence apparaît après le « creux » de $50ms$. En analysant plus finement les pics au début de ce creux, on constate que la divergence se produit à cet endroit.

En considérant ces dernières observations, nous pouvons affirmer que le *Header* non sécurisé est détecté dès les 45 premières millisecondes. Cela signifie qu'il est chargé, déchiffré et que le résultat de son premier test est donné durant ce laps de temps. Le déchiffrement du code en clair a produit l'effet inverse et a chiffré les valeurs du *Header*. La valeur `<0xBAAB>`, qui est sans doute un marqueur, n'a certainement pas pu être identifiée, provoquant ainsi l'échec de l'un des premiers tests. Une différence entre les processus des codes `BL_sec` et `BL_Hd.sec_IMG.nonsec` est observée au niveau des pics dans les « creux » de $50ms$. Hormis cela, les codes ont des comportements similaires pendant leurs premières $161ms$. Il y a de fortes probabilités pour que le déchiffrement et la vérification de l'image soient effectués dans ce laps de temps jusqu'à ce qu'un des tests échoue.

Sans approfondir la rétroconception, nous supposons que la première différence observée entre `BL_sec` et `BL_Hd.sec_IMG.nonsec` est induite par l'une des premières vérifications du code de l'image. Par conséquent, pour perturber le processus de *Secure Boot* juste après la validation du *Header*, nous définissons le temps de la génération de la perturbation électromagnétique t_{inj} légèrement avant cet instant.

4.1.3.2.6 Principe de l'attaque.

Le principe de l'attaque combinée est de charger un code capable de passer tous les tests de sécurité liés au *Header* puis de générer une perturbation avant le déchiffrement de la partie image. En effet, puisque le code est chargé en une fois mais déchiffré en deux étapes, nous supposons qu'il s'agit là d'une opportunité pour faire exécuter un code non chiffré.

Implémentation du code « malicieux »

Le code a été conçu pour passer la séquence de *Secure Boot* qui vérifie le *Header*. Pour cela, le *Header* d'un code sécurisé a été copié et mis en première partie de notre fichier qui va être chargé. Ce dernier renvoie une chaîne de caractères sur l'UART pour signaler qu'il

a bien été exécuté. La taille de notre fichier exécutable est inférieure à celle du code image qu'il remplace. Nous avons donc accolé une certaine quantité de '0x00' afin d'obtenir des codes de taille équivalente. Le schéma résumant l'implémentation de notre code est donné figure 4.11.

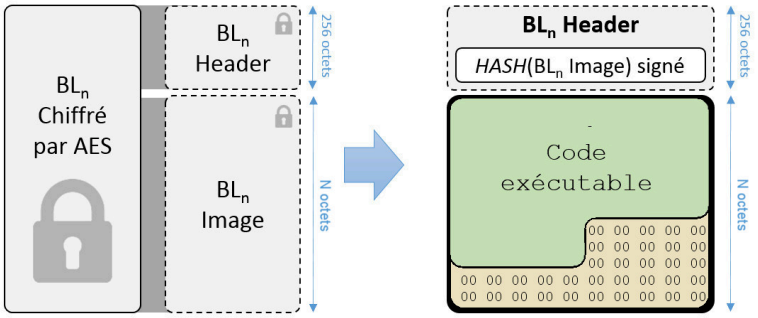


FIGURE 4.11 – Format du code « malicieux » utilisé dans notre attaque

4.1.3.2.7 Procédure d'injection de faute

La procédure consiste à pré-charger le code conçu pour notre attaque dans la NVM de la plateforme sécurisée. Puis, cette dernière est démarrée pour lancer la séquence de *Secure Boot* durant laquelle une perturbation électromagnétique est injectée. Si des données sont renvoyées sur l'UART, cela signifie que le code a été exécuté, sinon le SoC est en état `<mute>`.

L'ensemble de la surface de l'ASIC a été balayé avec le banc d'injection EM (voir Ch.2, Section 2.5). Pour chaque point spatial, un balayage temporel des injections EM a été effectué au niveau des pics détectés durant l'analyse *side-channel*. Ainsi t_{inj} s'incrémente progressivement de $t_{start} = 90,97ms$ à $t_{end} = 91,68ms$. Plusieurs cartographies ont été effectuées en faisant varier l'amplitude des injections électromagnétiques A_{inj}. Nous présentons les résultats les plus significatifs obtenus avec l'injecteur électromagnétique cylindrique de diamètre $850\mu m$ autour duquel 9 spires de cuivre sont enroulées (voir point (2), Section 2.5). L'amplitude et la durée d'injection sont respectivement configurées à $A_{inj} = -380V$ et $\Delta T_{inj} = 10ns$.

► Résultats expérimentaux

La surface considérée du SoC a été balayée en un peu plus de 2 jours. Sur les 700000 séquences de *Secure Boot* attaquées, 3 trames de données ont été retournées sur l'UART. Cela signifie que l'attaque a fonctionné pour 3 configurations des paramètres spatio-temporels. Pour synthétiser ces résultats, les trois attaques réussies se sont produites lorsque l'injection a été émise entre $91,040ms$ et $91,045ms$. Deux d'entre elles se situent au-dessus du même point spatial.

Amélioration du taux de réussite

Pour améliorer le taux de succès de l'attaque, nous avons placé l'injecteur EM au-dessus du point ayant produit les deux attaques réussies. Un nouveau balayage exclusivement temporel est effectué. L'instant de l'injection EM t_{inj} est incrémenté 500 fois de $t_{start} = 91,040ms$ à $t_{end} = 91,045ms$ avec un pas de $10ns$. L'attaque est répétée 10 fois pour chaque valeur de t_{inj}. Pour 5000 tentatives de perturbation, 39 ont été réussies, soit 0.78% des attaques, un taux de réussite multiplié par 1820.

4.1.3.2.8 Attaque exploitant une vulnérabilité d'implémentation du code de la ROM : conclusion

A l'aide de perturbation EM, nous avons montré qu'il est possible de charger et d'exécuter un programme non autorisé par la procédure de *Secure Boot*. De cette manière, il est possible de contourner la séquence de vérification et faire charger un code qui aura accès aux ressources matérielles. Ceci met en péril l'ensemble de la sécurité du système et permet à un attaquant d'imaginer divers scénarios d'attaques.

L'analyse de l'implémentation du *Secure Boot* a permis d'identifier des opérations « critiques » que nous avons exploitées dans la mise en place de nos perturbations. Ceci a permis de révéler l'existence d'une vulnérabilité conduisant au contournement du dispositif de sécurité. Nous supposons que l'origine de celle-ci est logicielle. La perturbation provoque soit un saut d'instruction soit un changement de valeur qui se traduit par un contournement de sécurité. Bien que l'efficacité de l'attaque soit inférieure à 1%, davantage d'analyses *side-channel* permettraient d'augmenter la précision. De plus, une unique attaque réussie, suffit en ce qui concerne les *Secure Boot*. Les concepteurs doivent prendre cela en compte et mettre en place des contre-mesures adéquates, par exemple de la redondance ou une vérification du flot d'exécution.

4.1.4 Etude de Faisabilité d'attaquer le Secure Boot : conclusion

Dans cette section, nous présentons les résumés d'élaborations d'attaques visant à évaluer la robustesse de *Secure Boot*. Cette présentation démontre qu'avec suffisamment de rétroconception et d'analyses *side-channel*, il est possible de localiser des points critiques qui peuvent contenir d'éventuelles failles exploitables. C'est par la sollicitation de ces points que des faiblesses ont été mises à jours. En l'occurrence, des vulnérabilités liées aux jeux d'instructions du processeur et à l'implémentation logicielle du *Secure Boot* ont montré la faisabilité de mettre en place des attaques par injection de fautes pour contourner cette sécurité.

4.2 ARM TrustZone®

Le TrustZone® est une extension de sécurité ajoutée aux processeurs ARM, qui leur permet de basculer entre deux états appelés mondes *secure* et *non-secure* . Les différentes ressources matérielles sont informées de l'état de sécurité dans lequel le processeur se trouve lorsqu'il les sollicite, rendant possible l'isolation complète d'un *monde* par rapport à l'autre.

4.2.1 Utilisation de TrustZone®

A partir des architectures ARMv6, l'extension de sécurité TrustZone®, a été introduite dans les processeurs ARM. Cette extension propose une gestion matérielle des contrôles d'accès qui permet de faire fonctionner deux processeurs virtuels isolés matériellement sur un même cœur. Ainsi pour protéger des données sensibles, un processeur applicatif va alterner son fonctionnement entre les deux processeurs virtuels. Dans la littérature, le *Secure World* (SW), désigne le processeur virtuel destiné à gérer les données sensibles et le *Normal World* (NW) désigne celui qui gère le reste. Lorsque le processeur physique est en mode *Secure*, cela signifie qu'il exécute des opérations liées au *Secure World*, tandis que le mode *NonSecure*, désigne l'exécution de processus du *Normal World*. La commutation entre les deux modes se fait de manière à ce que chaque *monde* puisse fonctionner indépendamment de l'autre. Les ressources telles que la mémoire ou les périphériques sont allouées selon le mode de fonctionnement du processeur. Les deux mondes ne sont pas symétriques, la distribution des ressources matérielles est effectuée pour que le monde *Secure* ait tous les accès, en particulier un accès exclusif à certaines ressources liées à la sécurité, tandis que le monde *NonSecure* a un accès restreint aux dispositifs nécessaires pour son fonctionnement. La propagation du niveau de sécurité dans lequel fonctionne le processeur est transmise à l'ensemble des périphériques du SoC via un signal dédié. Ce schéma rend inaccessibles les données du monde *Secure* à l'autre monde et permet d'exécuter des systèmes d'exploitation isolés physiquement dans chacun des *mondes*.

En règle générale, un système d'exploitation riche tel qu'Android [15] est exécuté dans le *Normal World*, tandis qu'un système d'exploitation minimaliste dédié à la sécurité (TEE) est exécuté dans le *Secure World*.

4.2.2 Principe de fonctionnement

Dans cette section, nous allons introduire les principes de fonctionnement du TrustZone® afin de discuter, dans la Section 4.2.4, des chemins d'attaques envisagés. La description détaillée de cette technologie est disponible dans les documentations [17, 137].

Le TrustZone® est un ensemble de solutions logicielles et matérielles qui permettent de faire fonctionner un processeur selon un niveau de sécurité *Secure* ou *NonSecure*. Afin d'implémenter ces solutions sur diverses architectures matérielles, selon les ressources disponibles et les besoins de sécurité, ARM met à disposition divers IP-bloc pouvant être combinés afin de garantir l'isolation d'un processus d'un bout à l'autre. Nous présentons dans cette section les bloc-IP nécessaires aux *Trusted Execution Environment* pour isoler physiquement l'OS *Secure* de l'OS *NonSecure*.

15. https://www.android.com/

4.2.2.1 Le processeur

Les caractéristiques des modes de fonctionnement des processeurs ARM sont rappelées dans l'Annexe A.2 de ce document. Avec l'architecture TrustZone®, le nouveau mode *Secure Monitor* administre les deux états de fonctionnements : *Secure World* et *Normal World*. Désormais un processeur TrustZone® dispose de 4 modes de fonctionnement :

- *Secure* - privilégié. C'est le mode dans lequel travaillent le *Secure Monitor* et le *kernel* de l'OS du *Secure World*.
- *Secure* - non privilégié. C'est le mode dans lequel travaillent la majorité des programmes du *Secure World*.
- *Non-Secure* - privilégié. Il s'agit du *kernel* d'un OS du *Normal World*. C'est le même niveau de privilèges que le *kernel* d'un OS fonctionnant sur un processeur n'implémentant pas l' extension TrustZone®.
- *Non-Secure* - non privilégié. Il s'agit d'un mode sans privilège. C'est le mode courant dans lequel se trouve le processeur la plupart du temps

La figure 4.12 schématise ces différents modes de fonctionnement ainsi que leurs privilèges.

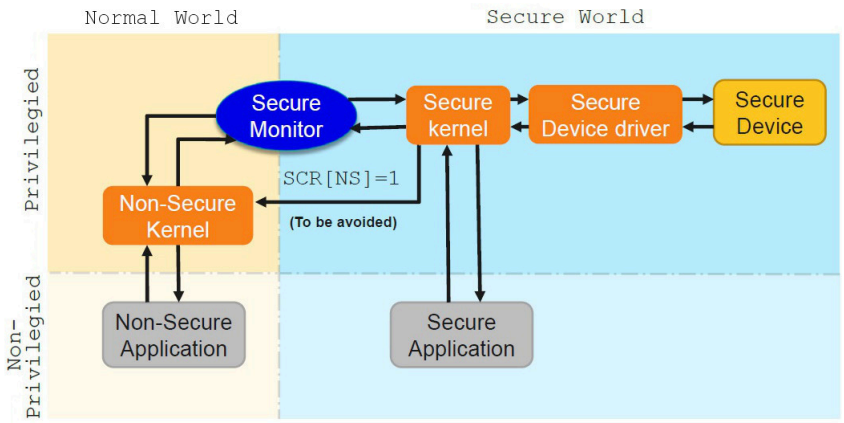
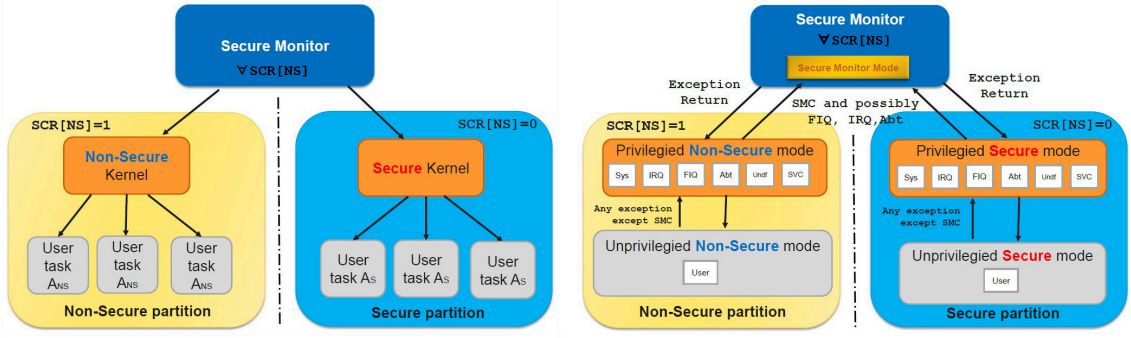


FIGURE 4.12 – Etat de fonctionnement des processeurs ARMv6 et ARMv7 TrustZone®

Deux registres sont utilisés pour définir l'état du processeur dans les différents modes :

- Le *Current Program Status Register* (`CPSR`).
- Le *Secure Configuration Register* (`SCR`).

Le `CPSR` contient le mode de fonctionnement (privilégié ou non privilégié). Le `SCR` contrôle le mode de sécurité (*Secure* ou *NonSecure*). Ce mode est définit par le bit `NS` (*NonSecure*) qui vaut '`1`', lorsque le système est en mode *NonSecure*, *Normal World* et qui vaut '`0`', lorsque le système est en mode *Secure*, *Secure World*. Ce registre est seulement accessible au mode privilégié du monde *Secure* et peut être utilisé pour modifier l'état du CPU à tout moment de manière asynchrone. Une tentative d'accéder à celui-ci par d'autres moyens provoque l'exception *Undefined*. La figure 4.13 illustre les modes de fonctionnements du CPU implémentant le TrustZone®.



a) Niveaux de privilèges b) Modes de fonctionnement

FIGURE 4.13 – Modes de fonctionnements des processeurs ARM intégrant l'extension de sécurité TrustZone®

4.2.2.2 Les bus

Les caractéristiques des différents bus de la 3ème version de la norme ARM AMBA sont rappelées dans l'Annexe A.5 de ce document.

► **Les bus AXI.** Le bus AMBA3 AXI est un élément fondamental dans l'implémentation d'une isolation de processus couvrant l'ensemble d'un système. Ce bus haute performance relit les composants du système entre eux et les informe du niveau de sécurité. Des signaux supplémentaires ont été ajoutés à celui-ci afin qu'il propage l'état de fonctionnement `SCR[NS]` du processeur à l'ensemble du système. Il s'agit de la modification la plus caractéristique du TrustZone® :

- `AWPROT[1]`, pour les opérations d'écriture – '`0`' pour *Secure* et '`1`' pour *NonSecure*.
- `ARPROT[1]`, pour les opérations de lecture – '`0`' pour *Secure* et '`1`' pour *NonSecure*.

Avec l'architecture maître-esclave, lorsque le champ `NS` d'un maître est à '`1`', cela implique qu'il ne peut pas accéder à des esclaves *Secure*. Une erreur `SLVERR` (*slave error*) ou `DECERR` (*decode error*) est générée en fonction de la configuration matérielle.

► **Les passerelles AXI-to-AHB.** Les bus AHB ne transportent pas d'information relatives au niveau de sécurité implémenté dans le TrustZone®. Pour permettre aux bus AXI et AHB de collaborer dans la gestion des ressources, deux modules distincts sont nécessaires. Une passerelle AXI-to-AHB gère les requêtes maître de l'AXI vers l'AHB esclave et une passerelle AHB-to-AXI gère les requêtes maîtres de l'AHB vers l'AXI esclave. Ces passerelles associent le signal `NS` transporté par l'AXI à tout les composants connectés à l'AHB. Du point de vue de l'AXI, cela implique que tous les éléments derrière la passerelle sont considérés comme *Secure* ou *NonSecure*. Il est recommandé de ne jamais placer plusieurs composants susceptibles d'être simultanément en mode *Secure* et *NonSecure* sur la même passerelle.

► **Les bus APB.** Dans un système ARM caractérisique, la plupart des périphériques sont connectés au bus APB. Ce bus est plus simple et moins consommateur que le bus principal

AXI. Cependant le protocole AMBA3 APB ne transporte pas d'information relative au niveau de sécurité implémenté dans le TrustZone®. Cette restriction est voulue pour que les anciennes architectures périphériques ARM restent compatibles avec cette version d'APB. A la place, le niveau de sécurité des périphériques est administré par le module AXI-to-APB Brigde. Celui-ci fournit une interface entre le bus haute fréquence AXI et le bus basse consommation APB.

► **Les passerelles AXI-to-APB.** La passerelle AXI-to-APB fournit une interface esclave pour l'AXI afin de gérer jusqu'à 16 périphériques connectés à un bus APB. Ce module contient un décodeur logique d'adresses qui aiguille les requêtes ciblées de l'AXI vers un périphérique de l'APB. Les niveaux de sécurité de chaque périphérique sont informés à la passerelle par autant de signaux `TZPCDECPROT`. La passerelle rejettera toute opération *NonSecure* ciblant un périphérique informé comme *Secure* dans son décodeur d'adresses.

► **Contrôleur TrustZone-Protection (TZPC).** Ce contrôleur est un module configurable qui peut être utilisé conjointement avec les passerelles AXI-to-APB pour que des périphériques *Secure* et *NonSecure* puissent partager un même bus APB. La figure 4.14 donne un exemple pour cet usage. Ce contrôleur comporte 3 registres `TZPCDECPROT`$\{2:0\}$, chacun capable de contrôler 8 signaux dans le SoC. Il contient aussi un registre `TZPCROSIZE` pouvant être utilisé pour fournir les signaux de contrôle des emplacements des partitions pour l'adaptateur de mémoire TrustZone® (voir TZDMA).

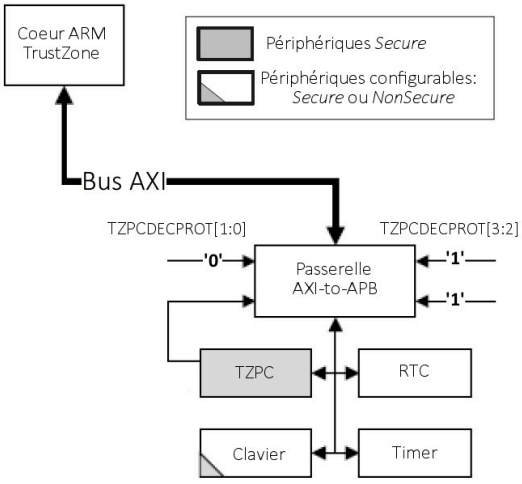


FIGURE 4.14 – TZPC utilisé avec un AXI-to-APB controllant 4 périphériques. Le TZPC est programmé comme étant toujours *Secure*, le Timer et le Real-Time-Counter (RTC) sont toujours *NonSecure*. Le clavier a un niveau de sécurité programmable. Durant sont exécution, le logiciel du *Secure World* peut programmer le TZPC au travers de la passerelle AXI-to-APB pour mettre le clavier en mode *Secure* ou *NonSecure*

4.2.2.3 Les mémoires

Afin de séparer les contextes de fonctionnement des différents éléments des SoC et de protéger les informations *Secure* du monde *NonSecure*, l'isolation des mémoires par les solutions TrustZone® est essentielle. L'ajout de l'information du bit `NS` à l'ensemble de la gestion des mémoires fait que le système peut être vu comme s'il comportait un bit supplémentaire dans la valeur de ses adresses. Avec l'extension TrustZone®, un système utilisant une gestion des adresses 32 bits pourra être considéré comme un système ayant un adressage sur 33 bits : 32 pour le monde *Secure* et 32 pour le monde *NonSecure*. Les caractéristiques des différentes mémoires utilisées dans les SoC sont rappelées dans l'Annexe A.3.

► **La mémoire principale.** La mémoire principale est physiquement séparée, réservant une région pour le *Normal World* et une autre pour le *Secure World*. Cependant, certaines zones mémoire peuvent être partagées entre les deux niveaux de sécurité. Toutes les adresses physiques sont vérifiées dans les processus de sécurité du TrustZone®. Cela implique que la MMU liste le niveau de sécurité de chaque région et que les partitions soit programmées dans l'interface AXI.

► **Contrôleur de cache niveau 2.** L'utilisation de mémoires cache de niveau 2 est nécessaire pour réduire la consommation énergétique et conserver les performances des systèmes. Cependant, lorsqu'un processus `NonSecure` succède à un processus *Secure*, les pages mémoires de données sensibles qui ont été stockées dans la cache L2 sont toujours disponibles. La première solution serait de vider entièrement la cache à chaque alternance d'état de sécurité. Or, en procédant de la sorte, les performances de rapidité de traitement s'effondrent. La solution implémentée dans TrustZone® est un contrôleur qui annote l'information de sécurité `NS` à chaque ligne contenue dans la cache.

► **Contrôleur DMA.** Les contrôleurs DMA sont des systèmes optimisés pour les transferts de données mémoire afin de décharger le processeur principal de ce genre de tâche. Dans le cadre du TrustZone®, ces contrôleurs sont conçus afin d'être capables de gérer des accès simultanés *Secure* et *NonSecure*, chacun avec des ensembles d'interruptions différentes. Ce sont des modules multi-canaux connectés à l'AXI et gérés par une interface APB.

► **Contrôleur d'espaces d'adresses TrustZone (TZASC).** Ce contrôleur est un composant AXI pour les mémoires extérieures au SoC. Il est principalement utilisé pour distinguer les adresses d'esclaves *Secure* de celles *NonSecure*. Les différentes plages d'adresses sont regroupées suivant un nombre de régions qu'un programme du *Secure World* peut configurer comme étant *Secure* ou *NonSecure*. Ce contrôleur ne fonctionne qu'avec les mémoires dynamiques (DRAM) et autorise les accès en rafale (*burst access*). Il ne permet pas de gérer les mémoires de traitement par bloc telles les flashs NAND.

► **Adaptateur mémoire TrustZone (TZMA).** Ce contrôleur est destiné à séparer les régions *Secure* des *NonSecure* dans les mémoires statiques internes au SoC (ex. ROM,

SRAM). Placer une mémoire conséquente et la partitionner en zones *Secure* et *NonSecure* est moins onéreux que de consacrer une mémoire pour chaque *monde*. Le TZMA partitionne une mémoire statique de $2Mo$ maximum en deux régions avec les premières adresses attribuées au *Secure World* et les dernières au *Normal World*. La capacité de la partition *Secure* est un multiple de $4ko$ contrôlée par le signal d'entrée `ROSIZE`. Cette dimension peut ainsi varier dynamiquement en reliant ce signal à la sortie `TZPCROSIZE` du contrôleur TZPC. Pour une taille fixe, l'architecture du système maintient le signal à une valeur donnée. Ce contrôleur ne peut pas être utilisé si plus d'une région *Secure* est nécessaire. Il ne fonctionne pas non plus sur les mémoires dynamiques.

4.2.2.4 Contrôleur d'interruption générique (GIC)

Pour supporter une gestion fiable des interruptions *Secure* ou *NonSecure*, il faut intégrer dans le processeur soit un GIC gérant les caractéristiques TrustZone® soit deux GIC, chacun dédié aux interruptions d'un des deux *mondes*. Le système de GIC intégré ne doit pas permettre au *Normal World* de modifier la configuration des interruptions du *Secure World*. Pour éviter toute attaque de déni de service, seules les interruptions de basses priorités doivent être configurables par les logiciels du *Normal World*.

4.2.2.5 Le débogage

L'isolation d'un système doit également prendre en compte les systèmes de débogage. Dans le chapitre 5, consacré au débogage matériel, nous décrivons les mécanismes de sécurité liés au débogage du TrustZone®. En particulier, la section 5.2.1 décrit comment, au travers de signaux de contrôle, l'isolation de contexte est maintenue, même durant une session de débogage.

4.2.3 Etude de faisabilité d'attaques du TrustZone®

ARM explique que sa technologie TrustZone® n'est pas conçue pour protéger contre des attaques matérielles effectuées avec des équipements de laboratoire. Néanmoins, peu d'études se prononcent sur la résistance de ce mécanisme alors que les utilisateurs ont besoin de pouvoir comparer les différentes solutions existantes.

4.2.3.1 Réflexion sur les vulnérabilités potentielles

Le TrustZone® est un ensemble matériel et logiciel dont la robustesse dépend de divers paramètres. Des vulnérabilités peuvent êtres introduites par les solutions technologiques utilisées pour implémenter les différents IP-bloc de cette technologie. Par exemple, une mauvaise interprétation ou une erreur de conception peuvent servir de chemin d'attaque. D'autre part, la technologie utilisée pour la fabrication des divers modules peut s'avérer plus ou moins vulnérable. Dans tout les cas, même si la théorie est certifiée robuste, seules des expérimentations peuvent conclure sur les vulnérabilités physiques d'une implémentation matérielle dans un système donné. Par conséquent, nos résultats seront dépendants des véhicules de tests.

Nous avons vu précédemment que l'isolation du mode de fonctionnement d'un processeur s'effectue selon deux états de sécurité. Cette isolation est assurée par divers modules dont la bonne coordination est une condition nécessaire pour la sécurité du système. Deux aspects sont évoqués ici : la propagation matérielle de l'état de sécurité à tous les éléments impliqués dans une même opération et la gestion logique des tâches. En considérant ces deux caractéristiques, notre étude a été scindée en deux étapes. Une première partie, effectuée dans cette section, cherche à estimer la capacité à perturber physiquement la propagation de niveau de sécurité dans le système. Pour cela, nous perturberons les modules impliqués dans les solutions de sécurité du TrustZone®. La seconde partie vise le système d'exploitation du *Secure World*. Cette étude, effectuée dans la Section 4.3, évalue la faisabilité d'attaquer la partie logique qui utilise les éléments du TrustZone®.

4.2.3.2 Principe des expérimentations

Comme décrit dans la Section 4.2.2, les modules impliqués dans la solution TrustZone® sont en grande partie des contrôleurs matériels dont la configuration est effectuée au travers de banques de registres. Depuis le processeur avec le registre de configuration de sécurité `SCR[NS]`, en passant par le bus AXI et ses signaux `A(R/W)PROT[1]`, jusqu'au contrôleur TZPC, l'information du niveau de sécurité est définie dans des registres où la valeur d'un bit change le niveau de sécurité. L'expérimentation effectuée ici vise à observer la capacité à modifier la valeur des registres d'un SoC.

4.2.3.3 Procédure d'injection de perturbation EM

Nous ne nous sommes pas restreints aux registres uniquement dédiés au TrustZone®. Pour un SoC donné, l'ensemble des registres configurables de manière logicielle a été ciblé. Le principe de la manipulation est de configurer l'ensemble des registres avec des valeurs de référence, de générer une perturbation électromagnétique puis de constater si une valeur a été modifiée.

Les expérimentations ont été effectuées sur le véhicule de test du Ch. 3 avec du code *bare-metal*. Ce SoC intègre 34 périphériques programmables, chacun configurable au travers d'une banque de registres 32 bits. Cela représente un nombre considérable de valeurs à observer (2038 registres répartis sur les 34 périphériques). Cependant, l'objectif du test n'est pas de précisément connaître la valeur d'une modification mais plutôt de vérifier s'il elle a bien eu lieu. Par conséquent, afin de simplifier l'analyse des observations, une somme de contrôle (*checksum*) sur 32 bit a été calculée sur la valeur de tous les registres contenus dans la banque de chacun des 34 contrôleurs. Afin de pouvoir synchroniser le déclenchement de l'impulsion électromagnétique, une commande « PULSE_EM » a été implémentée. Cette commande maintient le signal GPIO de la carte de développement au niveau « haut » pendant qu'elle exécute la répétition d'instructions '`NOP`'. Après cela, elle retourne la valeur de la variable qui va recevoir les différents *checksums*. L'impulsion est déclenchée durant les '`NOP`' et la carte de développement est redémarrée entre chaque impulsion EM.

L'ensemble de la surface représentant le silicium a été balayée (voir Fig.3.3 du Ch.3). Plusieurs configurations d'injections électromagnétiques ont été effectuées avec le banc d'injection EM (voir Ch.2, Section 2.5). Nous présentons ici les résultats les plus significatifs obtenus avec l'injecteur EM Cyl.-∅850-N9 et les paramètres décrits dans le tableau 4.2.

4.2.3.3.1 Résultats expérimentaux

Les $36.10^6 \mu m^2$ à la surface du SoC ont été balayés en approximativement 6 heures. Sur les 10201 injections, il y a eu 543 `<mute>` et 4462 *checksum* modifiés, soit respectivement $5,33\%$ et 43.74% des injections. Si, à première vue, ces résultats semblent traduire une grande sensibilité des registres face aux perturbations électromagnétiques, il est nécessaire d'analyser le type de contrôleur dans lesquels les modifications ont eu lieu pour pouvoir tirer des conclusions.

Paramètre	**Valeur**	
Zone balayée : $L = l$	6000 μm	} Matrice (101×101) ; 10201 essais
Pas : $\Delta p_x = \Delta p_y$	60 μm	
Nombre d'essais par point : N	1	
Amplitude de l'impulsion : A_{inj}	$+400V$	
Durée de l'impulsion : $\Delta_{T_{inj}}$	$6ns$	

Tableau 4.2 – Paramètres de balayage avec l'injecteur EM Cyl.-∅850-N9

Contrôleur	Nb. checksum modifiés
PLL	4235
PULSE_EM	108
Alimentations	15
Module de sécurité 1/2	14
UART 2	12
Power Gating CPU	10
USB	9
Horloges	5
I/O multiplexeur	4
`<reset>` du système	4
Digital Host	3
Module de sécurité 2/2	3
Adaptateur de données vidéos	2
Passerelle AHB-to-IP 1	2
Passerelle AHB-to-IP 2	2
DDR	2
Ethernet	2

Contrôleur	Nb. checksum modifiés
GPIO	2
Interface modules externes	2
Interface audio	2
Module traitement vidéo	2
Module de sécurité 3	2
Multiplexeur Audio	2
OTP	2
Périphériques partagés	2
Pulse Width Modulation	2
ROM	2
SPI	2
Synchronous Serial Interface	2
Timer interruptions	2
UART 1	2
USBphy	2
MIPI_CSI	0
MIPI_DSI	0

Tableau 4.3 – Liste des *checksums* sur la valeur des registres des contrôleurs modifiés.

Analyse des checksums modifiés

Pour analyser les changements de valeurs, nous allons nous baser sur le tableau 4.3 et la figure 4.15 qui synthétisent les modifications de registres durant cette campagne d'injections. Le tableau liste les contrôleurs dont la valeur de *checksum* a été altérée et la figure,

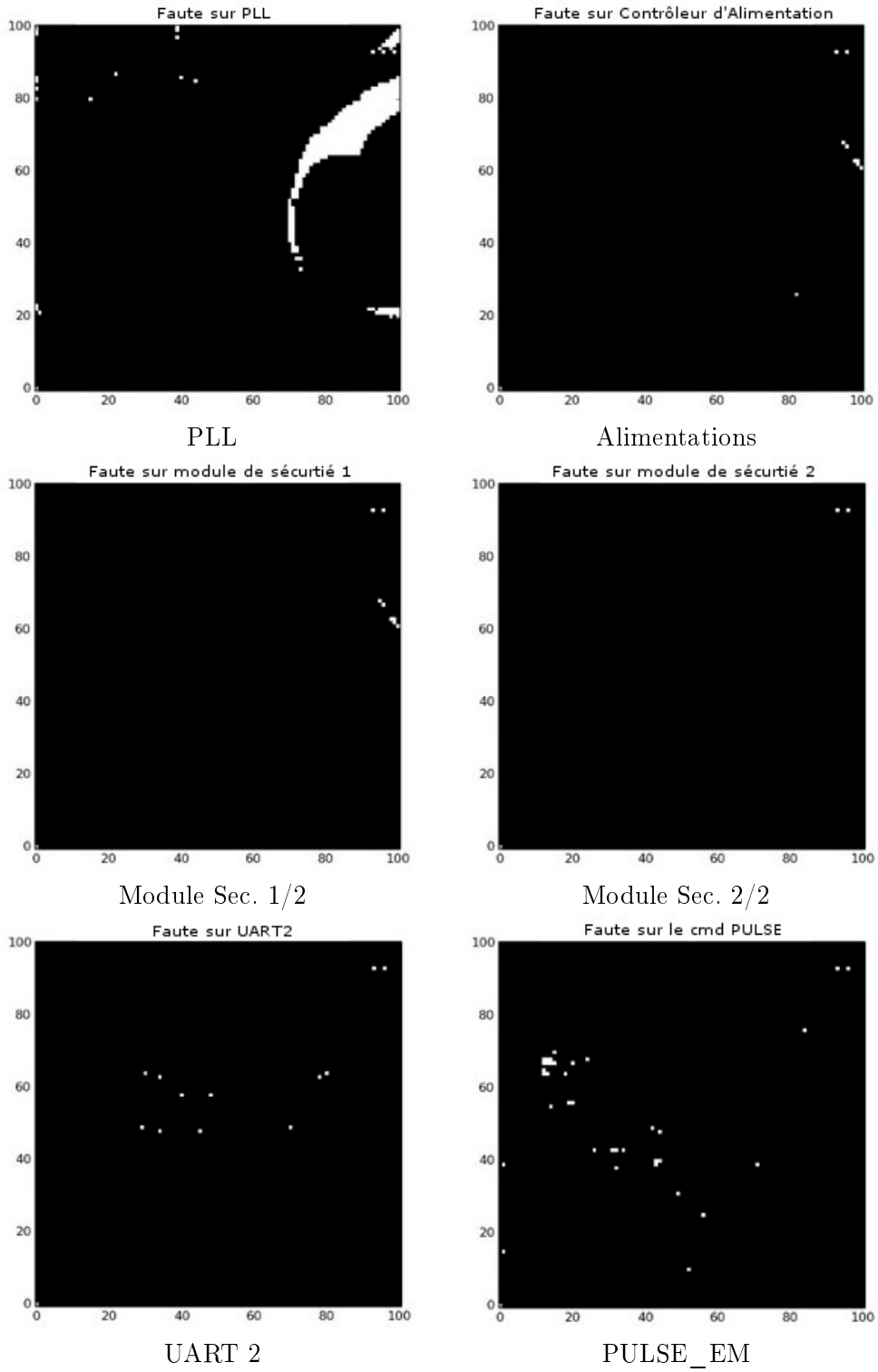


FIGURE 4.15 – Localisation spatiale des modifications de valeurs de registres pour les contrôleurs les plus sensibles

donne la localisation spatiale des modifications pour les contrôleurs les plus sensibles.

Sur 34 modules observés durant les perturbations électromagnétiques, 32 ont été modifiés. Deux modifications semblent être communes à plusieurs valeurs de *checksum*. Il s'agit des deux modifications représentées spatialement sur la Fig. 4.15 par les deux points côte-à-côte dans le coin supérieur droit de chaque cartographie. Il est envisageable qu'il s'agisse d'une information fournie par un détecteur de sécurité ou d'un artéfact induit par un comportement non envisagé du SoC.

Les registres ayant étés les plus impactés sont ceux liés aux PLL, ils représentent 41.51% des modifications. Dans une moindre mesure, on relève des changements sur les registres du contrôleur d'alimentation électrique, sur les registres d'une partie d'un module de sécurité, sur l'UART 2, sur le « Power Gating » du processeur, sur l'USB et le contrôleur d'horloges. En revanche, les registres du TrustZone®, comme ceux des TZPC inclus dans le *checksum* des contrôleurs « AHB-to-APB », ne semblent pas avoir été modifiés significativement. La valeur d'initialisation de la variable retournée par la commande « PULSE_EM » a été changés après 108 injections électromagnétiques. L'injection se faisant durant la répétions des '`NOP`', plusieurs explications sont possibles mais difficiles à confirmer comme la modification d'*opcode* par exemple.

La majorité des contrôleurs dont les registres ont été modifiés ont un rapport avec l'alimentation électrique ou les horloges. L'UART 2 est le port de communication utilisé pour l'échange de données avec le banc d'injection. Le fait que des registres liés à l'alimentation soient impactés permet de penser que ce ne sont pas directement les valeurs qui ont été modifiées mais plutôt que c'est le contrôleur qui ait modifié ses registres suite à la perturbation électromagnétique. Le comportement qui appuie cette supposition est celui des registres du contrôleur des PLL. La figure 4.15 illustre une zone étendue du SoC sur laquelle les registres de ce contrôleur ont été modifiés. Or ce dernier n'est composé que de 59 registres, et par conséquent, il est plus probable que les injections EM perturbent directement les PLL plutôt que les valeurs des registres. Les modifications observées sur le *module de sécurité* confirment également cette supposition. Il s'agit d'un module qui fournit une horloge de référence pour le système (*Real Time Counter*) et gère la clef maître. Il est équipé de détecteurs de *glicth* qui sont probablement responsables de la modification des registres afin d'informer l'ensemble du système d'une éventuelle attaque. Il serait possible de confirmer ces hypothèses en reproduisant cette manipulation et en allant lire directement la valeur de certains registres.

4.2.4 Etude de faisabilité d'attaques du TrustZone : conclusion

Dans cette expérimentation, nous avons voulu vérifier s'il était possible de modifier les valeurs contenues dans les registres de modules intégrés dans les SoC. Dans l'affirmative, cela aurait permis de valider un chemin d'attaque, puis il aurait fallu cibler les contrôleurs décrits dans la Section 4.2.2 pour montrer la possibilité d'une éventuelle attaque sur le TrustZone®. Or, les expérimentations effectuées dans cette section n'ont pas engendré de modifications significatives des valeurs contenues dans les registres. Il est possible que nous n'ayons pas réussi à paramétrer la configuration d'injection EM adéquate. Dans tous les cas, cela a permis de mettre en évidence les difficultés à modifier des valeurs déjà

établies dans une banque de registres de contrôle avec le canal EM et d'autre part, cela donne lieu d'écarter certains scénarios d'attaques. S'il fallait concevoir une attaque ciblant le TrustZone®, les vulnérabilités basées sur des modifications de valeurs de registres ne seraient pas les premiers chemins d'attaques à exploiter.

Les tests visaient des registres dont les valeurs étaient fixées avant la génération des perturbations EM. Il serait intéressant dans un travail ultérieur, d'approfondir ces expérimentations en ciblant non pas une valeur de registre statique mais plutôt une transition de valeur. Le fait qu'une valeur soit acheminée et non fixée implique un plus grand nombre d'éléments susceptibles d'être perturbés. L'horloge, les bus et les bascules du registre, sont tous des éléments qui peuvent être impactés par une impulsions EM [99].

4.3 Trusted Execution Environment (TEE)

Le *Trusted Execution Environment*, ou *environnement d'exécution de confiance* en français, est un système d'exploitation qui garanti, sur un même appareil, un contexte applicatif sécurisé et isolé des autres systèmes d'exploitation.

4.3.1 Utilisation du TEE

Avec la démocratisation des *smartphones* et autres systèmes du même type, le public a tendance à regrouper sur un même appareil l'ensemble de ses données avec diverses applications. Ainsi, des identifiants, mots de passe, contacts, e-mails, documents divers, photographies, jeux, multimédia, etc. évoluent et se côtoient au sein d'un même appareil. Pour les différents acteurs de ce modèle économique, cette pratique augmente le risque de compromettre certaines données sensibles.

Prenons l'exemple d'une personne qui utilise son téléphone portable pour se divertir et gérer sa vie personnelle et professionnelle. Il n'y a aucune garantie pour qu'une quelconque application téléchargée depuis une plateforme « magasin d'applications », ne dissimule pas des dispositifs capables d'extorquer certaines données présentes sur l'appareil. Faire évoluer, dans un même contexte applicatif, des documents d'entreprise, des œuvres soumises aux droits d'auteurs et des données personnelles, augmente la surface d'attaque des données sensibles. Pour répondre à cette problématique, deux solutions existent :

- Ne pas regrouper sur un même appareil toutes ses activités et documents.
- Mettre en place des dispositifs de sécurité robustes et adaptés à ce type de pratiques.

Par souci d'économie et parce que l'utilisateur est souvent la principale vulnérabilité d'un système, la deuxième solution est celle qui est adoptée sur les nouveaux modèles de *smartphones*. Ainsi des TEE tels que *Knox*[16], *Secure Enclave*[15] ou autres, ont été développés pour fonctionner conjointement avec Android ou iOS et garantir des environnements d'exécution de confiance. Ces mini OS fournissent des contextes isolés dans lesquels les données et les programmes qui s'exécutent sont vérifiés et validés. De cette manière les données d'une entreprise ne sont accessibles qu'à travers l'interface du TEE, avec la garantie qu'elles évoluent dans un environnement exempt de codes non validés et potentiellement dangereux.

4.3.2 Principe de fonctionnement

Comme pour les *Secure Boots*, les TEE du marché ont des implémentations différentes avec des nuances quant aux divers protocoles établis pour assurer une isolation et un environnement d'exécution de confiance. Cependant, l'ensemble de ces caractéristiques font actuellement l'objet d'une normalisation initiée par le groupe GlobalPlatform[17] (voir Fig. 4.16). Afin de placer le contexte de notre travail, nous ne donnons dans ce paragraphe que les principes de fonctionnement de base du TEE étudié. Celui-ci étant conforme aux spécifications GlobalPlatform, l'ensemble des fonctionnalités sont explicitées par le cahier

16. https://www.samsungknox.com/fr
17. https://www.globalplatform.org/aboutus.asp

des charges fourni par le groupe [61].

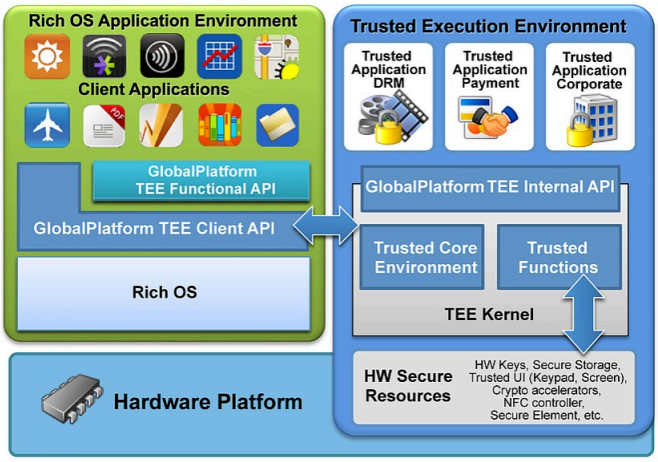


FIGURE 4.16 – Fonctionnement du TEE décrit dans les spécifications GlobalPlatform [18]

4.3.2.1 L'isolation des contextes

Afin de consolider la maîtrise de l'isolation qu'il fournit aux applications qu'il exécute, le TEE que nous étudions utilise les mécanismes de *Secure Boot* et TrustZone® décrits dans les sections précédentes. Le TEE est chargé sur l'appareil mobile par le *Secure Boot* qui lui assure la totale maîtrise des paramètres de sécurité du SoC. Cette situation lui permet de définir un contexte applicatif délimité et contrôlé, pour charger le système d'exploitation principal. Ce dernier, dénommé par contraste, le *Rich Execution Environment* (REE), est chargé, concluant ainsi le démarrage de l'appareil. Les deux OS fonctionnent de manière parallèle. L'utilisateur pourra à loisir télécharger des applications dans le REE et profiter des fonctionnalités offertes par son appareil. Les applications côté TEE sont dénommées les *Trusted Application* (TA), en français applications de confiance. Elles sont développées grâce à un SDK associé au TEE et bénéficient d'un ensemble d'API pour effectuer diverses tâches sécurisées telles que du stockage, de la génération de nombres aléatoires, du chiffrement, etc. Afin de pouvoir utiliser ces fonctionnalités, des *Client Application* (CA) s'exécutent dans le REE. Ces dernières disposent de points d'entrées vers le TEE sous forme d'API spécifiques qui leur permettent de solliciter les services des TA auxquelles elles sont associées. Des portions de mémoires partagées sont mises à disposition entre une CA et sa TA pour échanger d'éventuelles données à traiter. La figure 4.17 illustre ce fonctionnement.

18. Source : GlobalPlatform white paper, "The Trusted Execution Environment : Delivering Enhanced Security at a Lower Cost to the Mobile Market", février 2011.

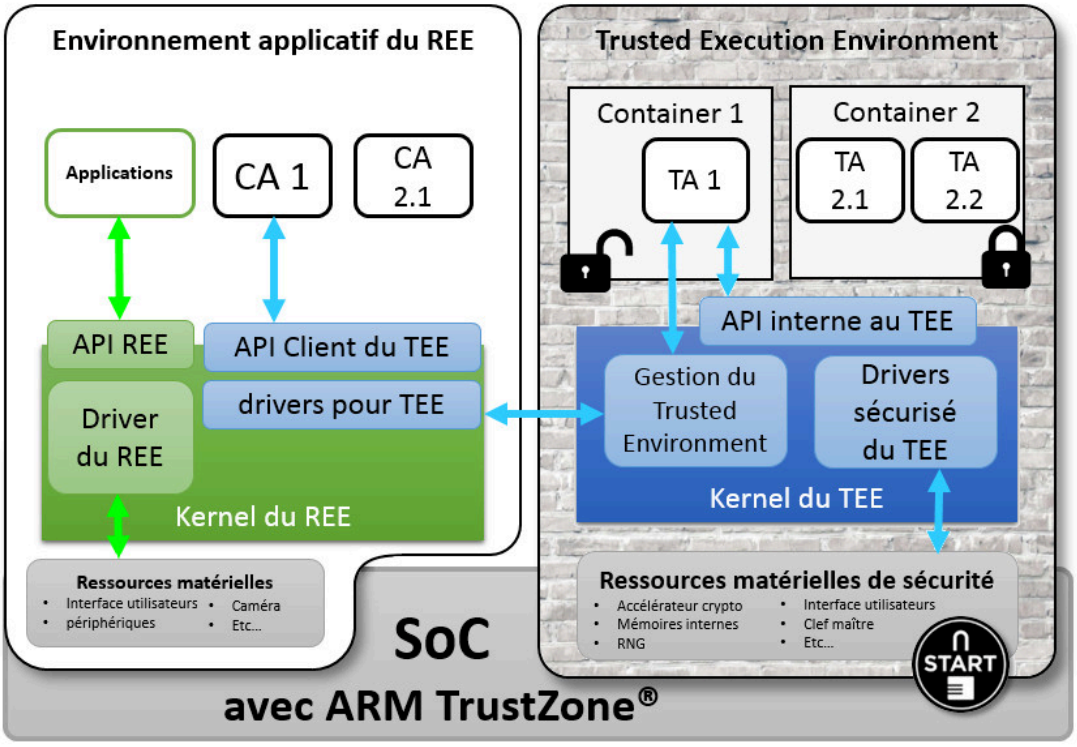


FIGURE 4.17 – Isolation des contextes applicatifs du TEE

4.3.2.2 Contrôle et cloisonnement des TA

Seules les TA autorisées peuvent être chargées dans un TEE. Comme illustré sur la figure 4.17, un système de container est mis en place de manière à cloisonner les espaces attribués aux TA provenant de différents fournisseurs de service (banques, entreprises, etc.). Un container est un espace chiffré qui contient toutes les TA provenant d'un même fournisseur de service. Une série d'accords préalables est passée entre le fournisseur de TEE, le concepteur de l'appareil sur lequel s'exécute le TEE, les fournisseurs de services et une entité de confiance : le *Trusted Application Manager*(TAM).

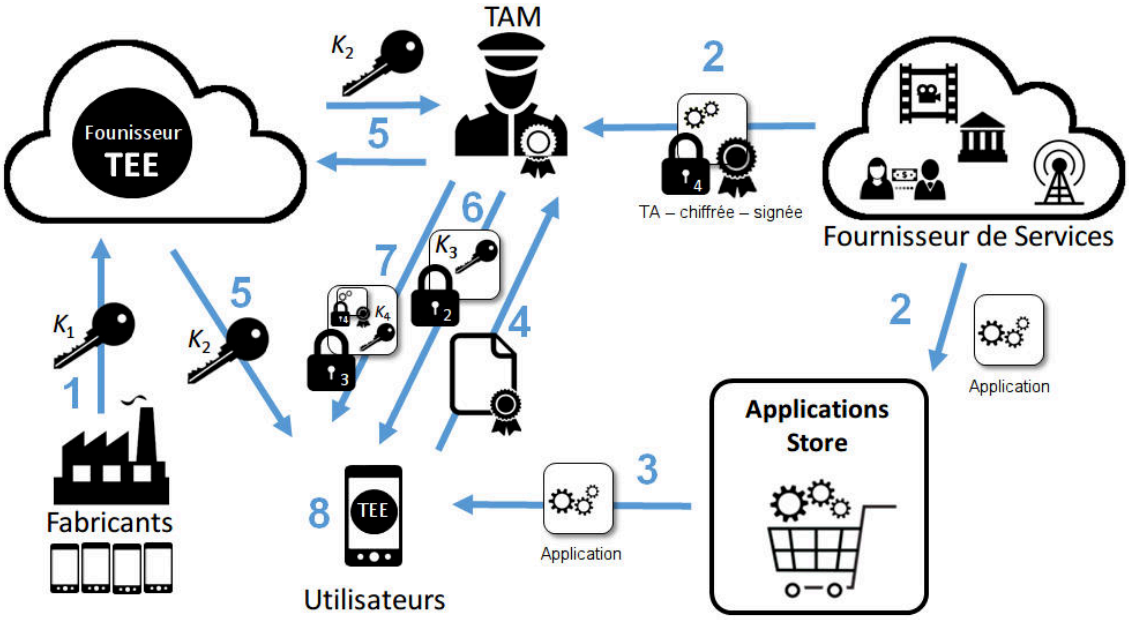


FIGURE 4.18 – Procédure d'activation et d'échange des clefs

La figure 4.18 décrit les processus impliqués dans l'activation d'un container et d'une TA liés à un fournisseur de service. La numérotation suivante correspond à celle de la figure :

1. Le fabricant d'appareils collabore avec le fournisseur de TEE et lui délivre une copie de la clef symétrique K_1. Cette clef est unique par SoC et sert à identifier un TEE sur un appareil auprès du fournisseur de TEE. Elle est stockée de manière sécurisée dans une mémoire persistante de l'appareil et sera utilisée pour dériver ou échanger d'autres clefs.

2. Un fournisseur de services développe d'un côté une TA et de l'autre une CA intégrée dans une application destinée au REE. Au préalable le fournisseur de service a conclu un accord avec le TAM qui lui a fourni une clef symétrique K_4 pour qu'il puisse transférer sa TA. Ainsi, le fournisseur de service transmet son application au magasin d'applications et la TA signée chiffrée au TAM ($C_{K_4}(TA)$).

3. L'utilisateur télécharge l'application du fournisseur de service sur le magasin d'applications, par exemple Google Play.

4. Lorsque cette application, qui contient la partie CA de la TA, arrive sur l'appareil contenant le TEE, celui-ci envoie une requête au TAM pré-sélectionné afin d'initialiser une procédure d'installation de la TA. Le TAM vérifie auprès du fournisseur de service s'il autorise l'installation de la TA en question sur l'appareil le sollicitant.

5. Le TAM demande au fournisseur de TEE la clef symétrique K_2 générée et envoyée simultanément de manière protégée au TAM et au TEE sur l'appareil. Ce dernier la conserve de manière sécurisée dans une mémoire persistante. Elle va servir pour activer un container et échanger une ultime clef.

6. La dernière clef symétrique K_3, générée par le TAM, est envoyée au TEE de l'utilisateur de manière chiffrée avec la clef K_2 : $C_{K_2}(K_3)$. Cette clef est conservée de manière permanente et est utilisée pour échanger des données avec le TAM concernant le container. De cette manière le fournisseur du TEE ne connaît pas la clef de chiffrement utilisée pour transférer les données au container.

7. Enfin, le TAM envoie au TEE de l'appareil, le chiffré de la TA et la clef K_4, le tout chiffré avec K_3 : $C_{K_3}(C_{K_4}(TA), K_4)$. Ils sont placés dans le container correspondant.

8. Le TEE utilise dans un premier temps K_3 pour déchiffrer $C_{K_3}(C_{K_4}(TA), K_4)$. Il place $C_{K_4}(TA)$ dans le container et K_4 dans un stockage sécurisé. Puis il utilise K_4 pour déchiffrer la TA et l'exécuter. Si, à l'avenir, le fournisseur de service souhaite placer une nouvelle *Trusted Application* dans le container qui lui est attribué, plus aucune clef ne sera envoyée à l'appareil. Le TAM enverra seulement la nouvelle TA chiffrée $C_{K_3}(C_{K_4}(TA'))$ dans le container correspondant.

4.3.3 Etude de faisabilité d'attaques d'un TEE

Parmi l'ensemble des fonctionnalités proposées par le TEE étudié, nous avons choisi d'analyser celles liées au *Secure Storage*, stockage sécurisé en français. En effet les TEE mettent en avant la protection des données par l'isolation des contextes d'exécution, il nous a donc semblé pertinent, dans le cadre de nos travaux, d'étudier la robustesse de celle-ci vis-à-vis d'attaques matérielles.

4.3.3.1 Véhicule de test - Contexte

Le TEE ciblé dans cette section est implanté sur une carte de développement intégrant un SoC octocœurs ARM-v8. Huit CPU ARM Cortex-A53 de 64 bits cadencés jusqu'à 1,2GHz exécutent Android v7.0 et le TEE étudié. Comme la majorité des plateformes de développement, cette carte fournit les interfaces nécessaires pour l'instrumentation de nos tests avec les bancs EM (GPIO, UART, etc.). Nous ne disposions d'aucune autre information sur ce SoC, exceptée celle disponible sur internet. De même pour le TEE, nous n'avions accès qu'aux documentations fournies aux développeurs pour utiliser le SDK et implémenter des TA et CA de test. Les expérimentations de cette section se sont donc faites en « boîte noire ».

4.3.3.2 Implémentation d'une Trusted Application pour nos tests

4.3.3.2.1 Le Secure Storage

Une application a besoin de lire et stocker des données. Une *Trusted Application* n'est pas différente. Le TEE étudié permet un stockage persistant et sécurisé des données. En utilisant des API similaires aux classiques `fopen`, `fclose`, `read`, `write` et `seek`, une TA peut stocker des données dans des fichiers-objets de manière sécurisée. Les identifiants et les contenus des fichiers sécurisés sont conservés de manière chiffrée dans la mémoire persistante du système. Ainsi, la confidentialité des données est protégée des processus non autorisés ayant des privilèges sur la mémoire persistante. L'intégrité des données est vérifiée avant chaque déchiffrement afin de prévenir toute modification par un processus non autorisé. Le cas échéant, la modification est détectée et reportée à la TA via un code d'erreur : `TEE_ERROR_CORRUPT_OBJECT`.

Ainsi, un développeur d'application a à sa disposition un espace de stockage sécurisé afin de pouvoir y entreposer des clefs cryptographiques, des configurations sensibles ou des données privées d'utilisateurs.

4.3.3.2.2 Implémentation

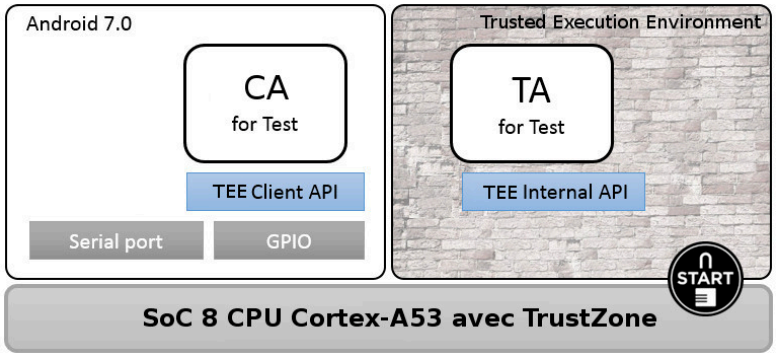


FIGURE 4.19 – Schéma de la *Trusted Application* développée pour les tests.

Les *Trusted Applications* développées grâce aux SDK du TEE sont composées de deux parties. La partie CA est le code qui s'exécute dans le REE. Elle est utilisée pour communiquer avec la seconde partie de code qui s'exécute dans le *Secure World* via les API fournies avec le TEE. Cette seconde partie de code est la TA à proprement parler, et ne peut être sollicitée que par le CA.

Afin de procéder à des expérimentations, nous avons dû développer des TA permettant de synchroniser les appareils qui composent les bancs de tests tels que des oscilloscopes, des générateurs d'impulsions EM ou le Raspberry Pi (décrits dans le chapitre 2). C'est pourquoi nous avons développé une application avec une partie CA qui gère la communication UART et les signaux GPIO de la carte de développement, et une partie TA qui travaille avec les

API du *Secure Storage* (voir Fig. 4.19). Les GPIO, uniquement contôlables depuis le REE, sont utilisés pour délimiter l'exécution de la TA dans le temps et déclencher les mesures de l'oscilloscope ou les impulsions du générateur. L'UART permet d'échanger des données ou des commandes entre les bancs d'évaluation et la TA.

La partie CA for Test

La partie CA sert d'interface entre le *Secure World* et le monde extérieur incluant le *Normal World*. Dans notre TA, ce code fonctionne comme un serveur dans l'environnement d'Android, écoutant le port série et attendant n'importe quelle instruction ou donnée pour la traiter. Les principales fonctions exécutées par cette partie de code peuvent être décrites par le diagramme 4.20.

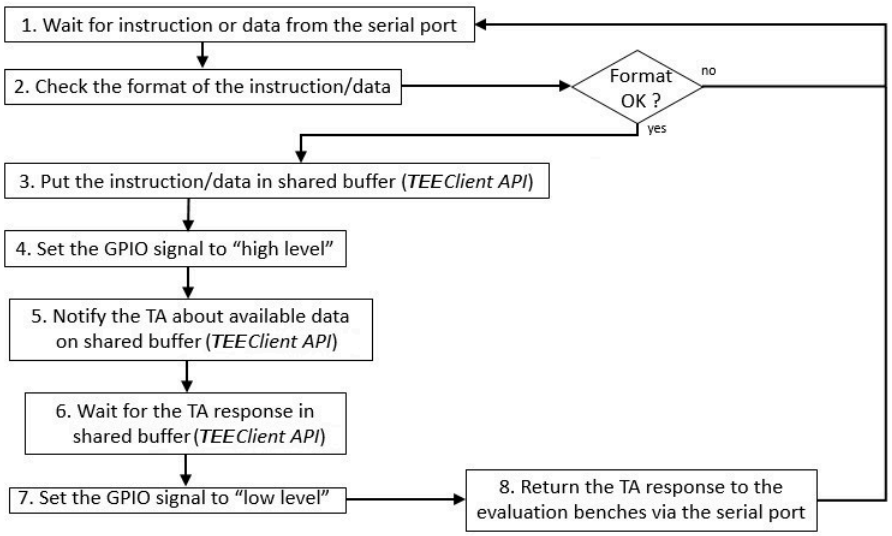


FIGURE 4.20 – Diagramme simplifié de la partie CA

La partie TA for Test

La partie TA est la partie du code qui s'exécute dans le *Secure World* et peut disposer des ressources matérielles sécurisées de celui-ci. L'utilisation de ces ressources s'effectue grâce à des API fournies par le TEE, dont une partie de ces dernières sont la cible de nos tests. La TA fonctionne également comme un serveur dans le SW et attend des instructions ou des données provenant du CA à travers différentes API spécifiques. Ces instructions ou données sont échangées via un système de *buffer* partagé entre les deux mondes. Les principales opérations exécutées par cette partie de code sont représentées sur le diagramme de la figure 4.21.

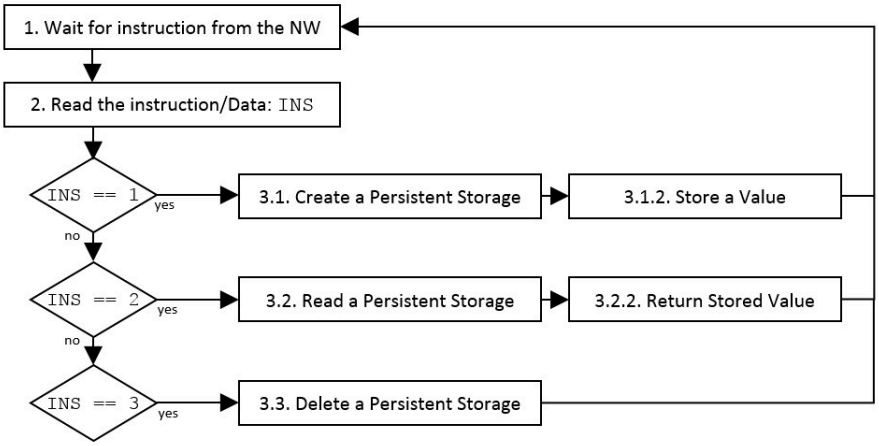


FIGURE 4.21 – Diagramme simplifié de la partie TA

Afin d'expliquer certaines contraintes que nous avons dû considérer dans nos expérimentations, nous allons détailler davantage certaines fonctions.

3.1. Create a Persistent Storage

Cette étape utilise l'API `TEE_CreatePersistentObject()` qui crée un *Secure Storage*, *i.e.* container sécurisé dans la mémoire non-volatile. Plusieurs arguments sont nécessaires dont notamment l'`objectID`, qui est la valeur permettant d'identifier un container. `Data` et `dataLen` représentent respectivement les données à stocker sous forme de tableau d'octets et la taille de celui-ci. Après la création d'un *Secure Storage*, la fonction `TEE_CloseObject()` est appelée pour fermer le container ainsi crée.

3.2. Read a Persistent Storage

Ce processus implique l'utilisation des API `TEE_OpenPersistentObject()` et `TEE_ReadObjectData()`. Pour lire les données d'un container sécurisé particulier, il est nécessaire d'accéder à celles-ci en « ouvrant » le container. Ceci est fait grâce à l'API `TEE_OpenPersistentObject()` auquel l'identifiant `objectID` du container est passé en argument. Ensuite, la fonction `TEE_ReadObjectData()` copie les données stockées depuis le container vers un *buffer* spécifié en argument. Enfin, la fonction `TEE_CloseObject()` referme le container sécurisé.

3.3. Delete a Persistent Storage

Cette étape utilise les API `TEE_OpenPersistentObject()` et `TEE_CloseAndDeletePersitentObject()`. Comme pour les autres étapes, la fonction `TEE_OpenPersistentObject()` est utilisée pour accéder à un *Secure Storage* et travailler avec. La fonction `TEE_CloseAndDeletePersitentObject()` le referme et le supprime définitivement.

4.3.3.3 Processus évalué

Nous avons choisi d'évaluer la résistance de la fonction « 3.2. Read a Persistent Storage » face à des perturbations électromagnétiques. Nous souhaitons vérifier l'isolation des différents containers et observer si le fait de perturber le processus de lecture des données sécurisées ne permettrait pas d'accéder à des données normalement inaccessibles. Afin de comprendre les résultats expérimentaux, la figure 4.22 détaille les API de la partie du code attaquée.

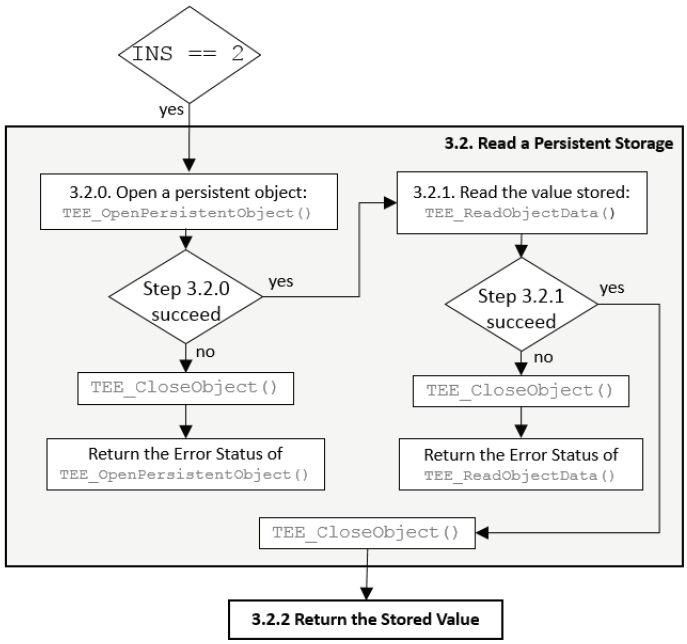


FIGURE 4.22 – Diagramme d'exécution de la fonction « 3.2. Read a Persistent Storage » de la Figure 4.21

4.3.4 Analyse side-channel

Les mesures *side-channel* on été effectuées avec le banc de mesure EM décrit dans la section 2.4 du chapitre 2. La sonde utilsée est une LANGER ICR HH 150-27 avec une bande passante de 6 GHz.

4.3.4.1 Localisation temporelle

Etant donné que nous avions la possibilité d'utiliser le SDK du TEE pour implémenter nos propres TA, nous avons également pu implémenter certaines parties de codes qui faciliteraient la localisation temporelle du processus ciblé. Comme expliqué dans notre méthodologie (Section 2.3.1.2), en plus des signaux GPIO utilisés pour borner le temps d'exécution de la TA, nous avons inclus, dans le code de la partie TA, une fonction susceptible de générer des variations de champs EM facilement identifiables. En effet, par mesure

de sécurité, le *Secure World* restreint au maximum ses interactions avec le monde extérieur. Ainsi il ne permet pas de générer un signal sur une I/O matérielle comme par exemple un port série ou un GPIO empêchant par la même occasion de synchroniser nos manipulations. Durant la recherche d'opérations générant une signature EM facilement identifiable, seule la multiplication a produit des motifs significatifs sur les différentes traces mesurées. Afin de s'affranchir des effets de caches et d'autres processus éventuels susceptibles d'occulter le travail de calcul CPU, des nombres aléatoires fournis par l'API `tlApiRandomGenerateData()` ont été utilisés. Par conséquent, nous avons défini la fonction `MarkerLoop()` qui génère des variations remarquables de champs EM que l'ont peut utiliser pour repérer des processus ciblés. Cette fonction prend deux octets (`P1`,`P2`) en arguments et exécute `P1` multiplications de nombres aléatoires de taille `P2` octets. Le code de cette fonction est donné par la figure 4.23.

```
1  int MarkerLoop(unsigned char P1, unsigned char P2)
2  {
3    int i = 0;
4    int j = 0;
5    uint8_t Buffer[2048] = {0};
6    size_t randomLen[1] = {0};
7    int tmpVar = 1; //to avoid 0*N = 0...
8    tlApiResult_t tlRet = TLAPI_OK;
9
10   //RANDOM multiplication :
11   while(i<P1)
12   {
13      randomLen[0]= P2;
14      tlRet = tlApiRandomGenerateData(TLAPI_ALG_SECURE_RANDOM,Buffer,randomLen);
15      if(TLAPI_OK != tlRet)
16      {
17         return tlRet;
18      }
19      for(j=0;j<P2;j++)
20      {
21         tmpVar *= Buffer[j];
22      }
23      i++;
24   }
25   return tlRet;
26 }
27
```

FIGURE 4.23 – Code de la fonction `MarkerLoop()`.

4.3.4.2 Utilisation de la fonction `MarkerLoop()`

Nous cherchons à identifier les signaux EM reflétant les activités du processeur travaillant pour le *Secure World*. Pour cela uniquement, la fonction `MarkerLoop()` est exécutée avec différentes valeurs pour le paramètre `P1`. Pour ces premières expérimentations, le balayage de la surface du SoC est effectué manuellement. Le signal GPIO permet de restreindre la durée des mesures acquises avec l'oscilloscope. Les mesures les plus significatives sont obtenues au dessus du point spatial montré sur la figure 4.24. La figure 4.25 présente les mesures de l'exécution de la fonction `MakerLoop()` avec trois ensembles `(P1,P2)` de valeurs différentes. Sur celle-ci, nous pouvons observer l'apparition d'un nombre de pics égal à la valeur de `P1`. Par conséquent, les motifs observables sur les mesures EM en ce point sont dépendantes des opérations effectuées. Nous nous placerons donc à cet endroit pour effectuer les prochaines mesures.

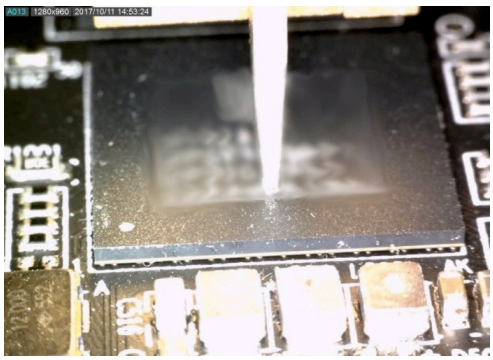


FIGURE 4.24 – Position de la sonde EM pour les mesures de signaux les plus significatifs.

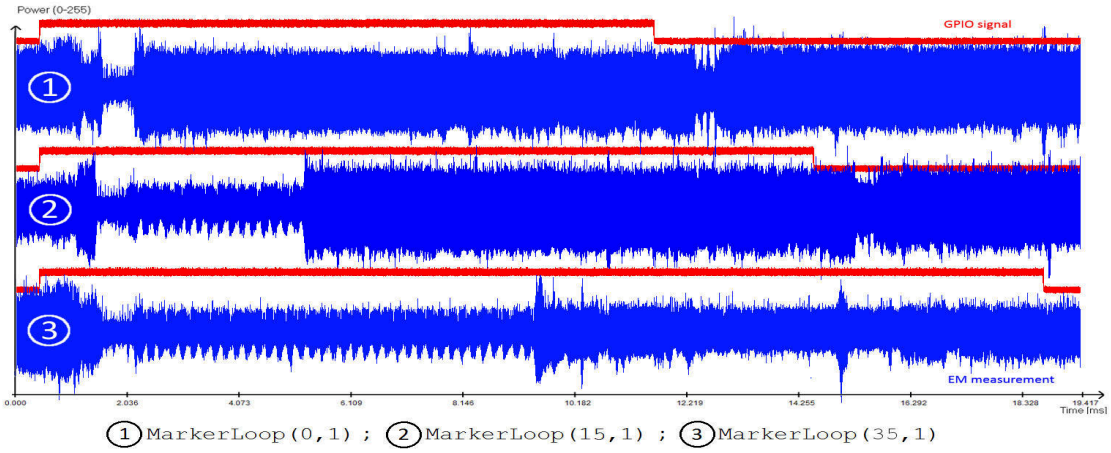


FIGURE 4.25 – Mesures des émissions EM de la fonction `MarkerLoop()` au point spatial de la Fig.4.24 avec différentes valeurs `(P1,P2)`

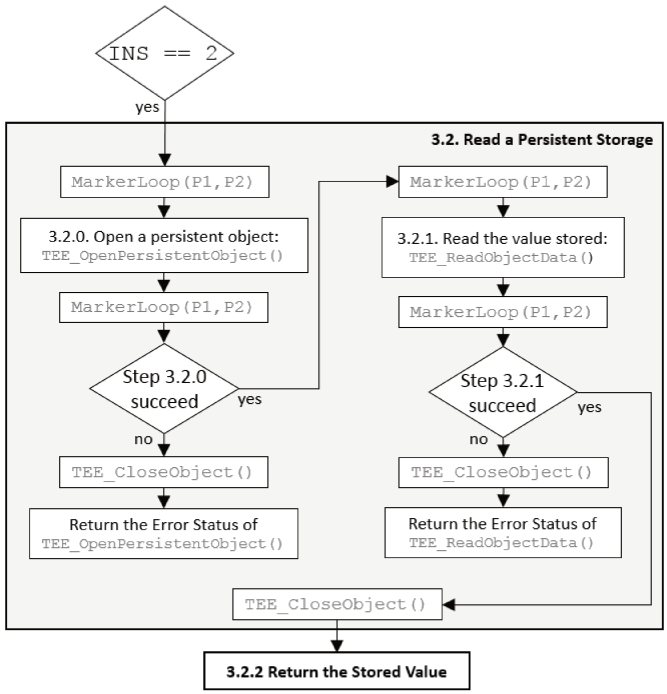


FIGURE 4.26 – Diagramme d'exécution de la fonction « 3.2. Read a Persistent Storage » avec les appels à la fonction `MarkerLoop()`

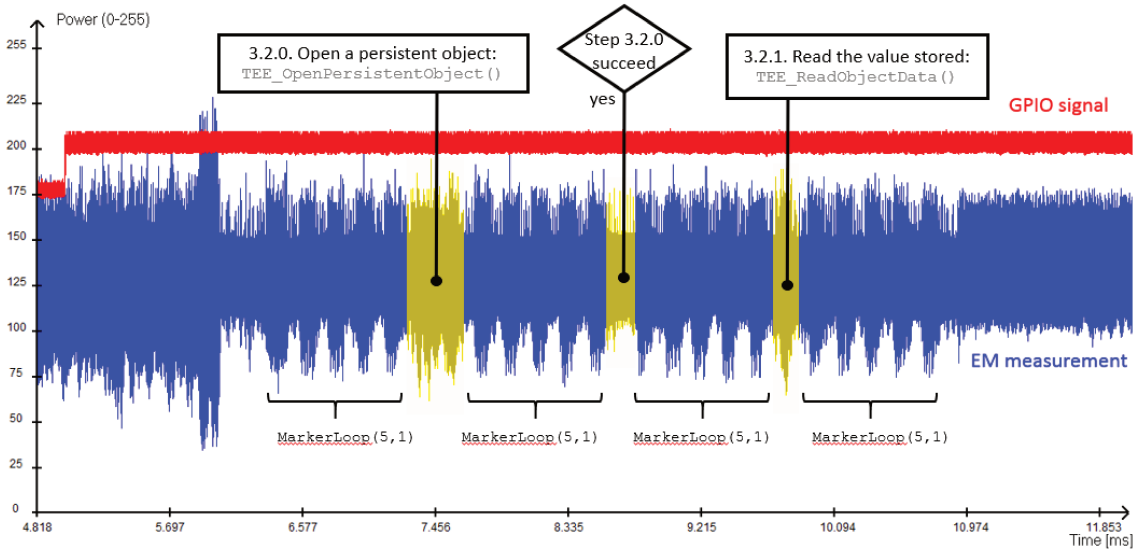


FIGURE 4.27 – Informations obtenues en considérant le diagramme de la Fig. 4.26

4.3.4.3 Localisation des appels de fonctions

Afin d'identifier les différentes fonctions API s'exécutant au sein des traces mesurées, plusieurs appels à la fonction `MarkerLoop()` ont étés ajoutés au code de la *TA*. La figure 4.26 montre l'addition des ces différents appels. La fonction « 3.2. Read a Persistent Storage » de la *TA* est exécutée avec les paramètres `P1=5` et `P2=1` pour la fonction `MarkerLoop()` et la mesure est effectuée au point de la figure 4.24. La figure 4.27 résume les informations obtenues grâce à cette manipulation. Ces mesures confirment le fait que le point spatial choisi reflète l'activité du processeur durant les opérations effectuées coté *Secure World*. Elles nous permettent également de connaître un paramètre primordial pour l'injection de fautes EM : l'instant de chaque opération.

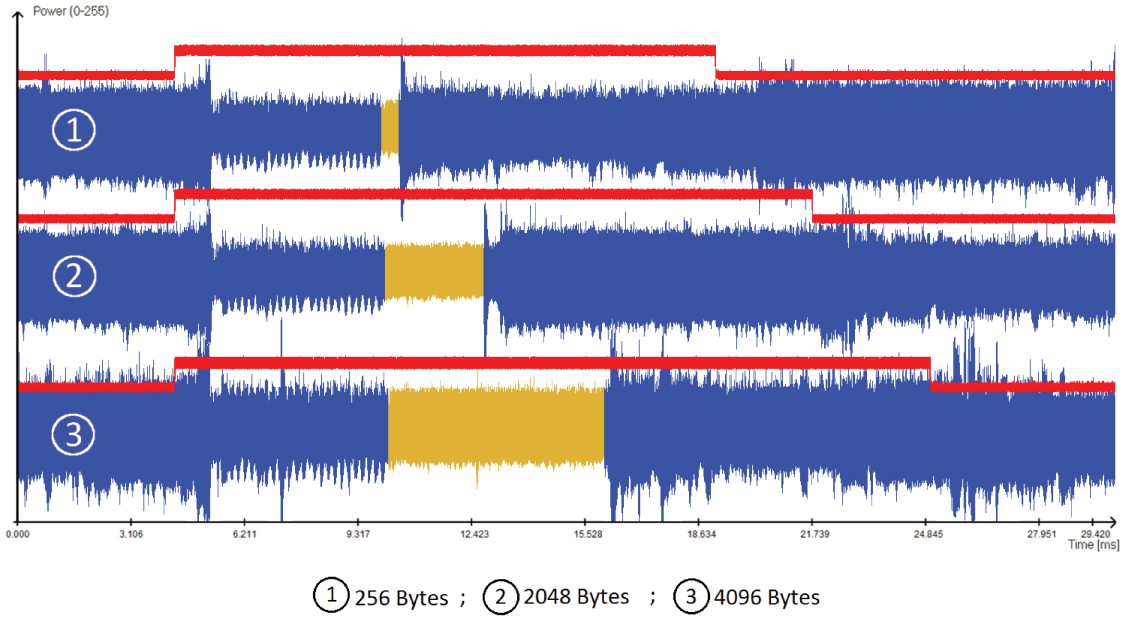


FIGURE 4.28 – Exécution du processus « 3.2. Read a Persistent Storage » sur des *Secure Storage* de différentes tailles. Les `memcpy` sont repérés en jaune.

4.3.4.4 Localisation de la copie des données.

Travaillant en boîte noire, nous ne connaissons pas le code des API exécutées par le TEE. C'est pourquoi nous essayons, par diverses manipulations, de connaître les détails des opérations effectuées par ces API. Ici nous cherchons à identifier un processus de recopie des données contenues dans les *Secure Storage* vers le *buffer* de sortie passé en argument à la fonction `TEE_ReadObjectData()`. Pour cela, trois *Secure Storage* de tailles différentes ont été créés : (1) 256 octets, (2) 2048 octets et (3) 4096 octets. Les mesures EM respectives sont données par la figure 4.28 sur laquelle nous observons deux caractéristiques. En premier lieu, la partie coloriée en jaune n'est pas liée à la recopie des données sécurisées par la fonction `TEE_ReadObjectData()`. Elle correspond au processus d'une fonction `memcpy()`

que nous avons utilisé dans notre code pour copier les données du *buffer* de sortie passées en arguments à la fonction API vers le *buffer* partagé entre les deux mondes. En effet, nous avons implémenté un *buffer* de sortie temporaire dans notre code au lieu d'indiquer directement le *buffer* partagé. Si tel était le cas, les parties jaunes présentes sur les traces n'existeraient pas. La seconde caractéristique observée sur le fonctionnement de l'API `TEE_ReadObjectData()` est que quelle que soit la taille des données manipulées, la durée d'exécution de cette fonction est la même. En effet, la partie bleue précédant les zones en jaunes est identique dans les trois cas.

4.3.4.4.1 Analyse side-channel : conclusion. Cette analyse *side-channel* nous a permis d'identifier les instants durant lesquels les différents API que nous souhaitons évaluer sont appelés. Nous avons ainsi la possibilité de définir l'instant auquel nous devons injecter une perturbation pour procéder à une attaque par faute. Nous avons également observé que la durée d'exécution de la fonction est la même quelle que soit la taille des *Secure Storage* alors qu'un `memcpy` est aisément identifiable.

4.3.5 Attaque Fault Injection

Les perturbations électromagnétiques ont été effectuées avec le banc d'injection EM décrit dans la section 2.5 du chapitre 2. L'injecteur EM Cyl.-$\varnothing$1000 – N6 est utilisé pour créer les perturbations à la surface du SoC.

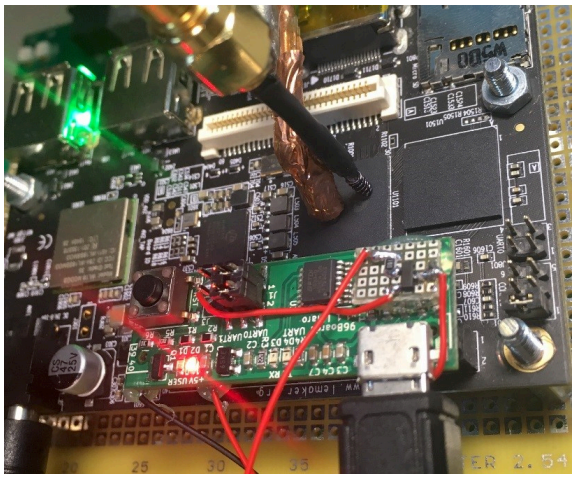


FIGURE 4.29 – Injecteur et sonde de mesure EM sur le DUT

4.3.5.1 Utilisation de la TA de test

L'objectif de cette expérimentation est d'évaluer la faisabilité de perturber la fonction « 3.2 Read a Persistent Storage », analysée dans la section précédente. Pour cela, 256 *Persistent Storage* de 4096 octets chacun sont créés (voir Tableau 4.4). Le script de test répète la lecture des données du *Persistent Storage* ayant l'ID = `0x80`. Une perturbation EM est générée durant cette lecture.

Persitent Storage	
ID	valeurs (4096 octets)
0x00	[0x00, 0x00, 0x00, ... , 0x00]
⋮	⋮
0x80	[0x80, 0x80, 0x80, ... , 0x80]
⋮	⋮
0xFF	[0xFF, 0xFF, 0xFF, ... , 0xFF]

Tableau 4.4 – *Persistent Storage* créés pour nos expérimentations

4.3.5.2 Détection des points sensibles

Afin de localiser les zones du SoC pour lesquelles le processus exécuté est sensible aux impulsions EM, toute la partie représentée par le quadrillage sur la figure 4.30 est considérée. Cette zone de $36.10^6 \mu m^2$ est balayée avec un pas de $60\mu m$ dans les deux directions $\vec{u_x}$ et $\vec{u_y}$. Le générateur d'impulsions est configuré pour produire des impulsions de $\Delta A_{inj} = +400V$ d'amplitude et $\Delta T_{inj} = 6ns$ de durée. Au-dessus de chaque point spatial , 10 essais d'impulsions sont effectués ciblant le processus du `memcpy()` à la fin de la fonction « 3.2 Read a Persistent Storage ».

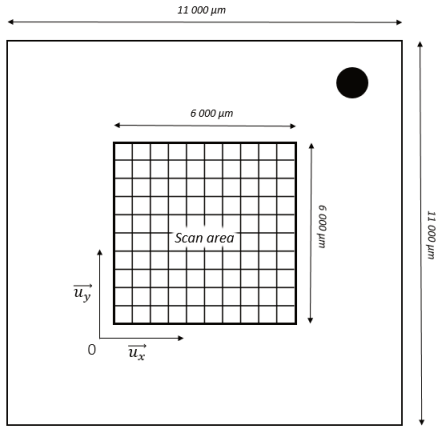


FIGURE 4.30 – Zone considérée à la surface du SoC

La synthèse de ce balayage est représentée par une matrice de 101 × 101 points spatiaux, avec 10 injections EM pour chacun d'entre eux. Cela représente 102010 essais dont les résultats sont résumés sur la figure 4.31. Sur tous les essais, 1980 ont généré un `<mute>` soit approximativement 2%. Aucune erreur de lecture n'a été relevée. Tous les points sensibles produisant des `<mute>` se situent dans la partie inférieure de la zone balayée. On en distingue deux sur la ligne $y = 8$, qui produisent plus de 8 `<mute>` sur 10 essais (un point en jaune et l'autre en blanc). Il s'agit de deux points très sensibles au-dessus desquels une impulsion EM perturbe quasi systématiquement le processus « 3.2 Read a Persistent

Storage ». Dans les prochaines manipulations, l'injecteur EM sera fixé au dessus du point $(81, 7)$ en blanc sur la figure.

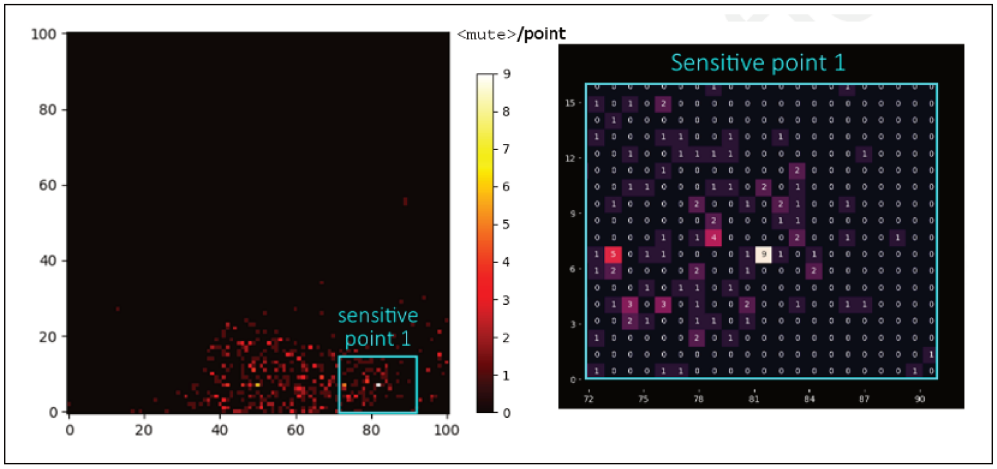


FIGURE 4.31 – Détection des points sensibles

4.3.5.3 Expérimentations avec une seule TA

En se basant sur les informations obtenues avec l'analyse *side-channel*, nous avons choisi d'étudier la perturbation de deux processus du *Secure Storage*. Le premier est la boucle de copie des données, située dans le code *Secure*, à la fin des diverses opérations et identifiée comme étant l'opération `memcpy()` dans la section 4.3.4.4. Le second processus est l'appel de la fonction `TEE_ReadObjectData()`. Ces deux expérimentations et leurs résultats sont donnés dans les sections qui suivent.

4.3.5.3.1 Perturbation de la fonction `memcpy()`.

Dans cette partie , l'étude vise à évaluer la possibilité d'avoir une vulnérabilité dans une fonction *simple* s'exécutant dans l'OS sécurisé. La fonction ciblée ne fait pas partie des API du TEE mais du code de la TA. Nous supposons que la perturbation de la fonction `memcpy()` travaillant avec des pointeurs d'adresses peut générer la copie d'autres données, sécurisées ou pas, de manière incontrôlée par le TEE.

Procédure d'injection EM.

Le logiciel présent sur le PC de contrôle du banc de mesure électromagnétique (voir (5), Section 2.4 du Ch.2), a une interface qui limite la taille des données échangées à 256 octets. Cette taille suffit pour les expérimentations pour lesquelles le banc a été dimensionné. Par conséquent, afin de détecter d'éventuelles modifications dans les données copiées, le balayage temporel effectué dans cette expérimentation est localisé au début de la fonction `memcpy()`. Sur une fenêtre temporelle de $\Delta T = 12\mu s$, huit essais de perturbation sont

effectués toutes les $1ns$ durant la copie des données du *Persistent Storage* n° `0x80` (voir Fig.4.32). Seuls les 256 premiers octets sont retournés. Chaque impulsion EM générée a une amplitude $A_{inj} = +321V$ et une durée de $\Delta T_{inj} = 6ns$.

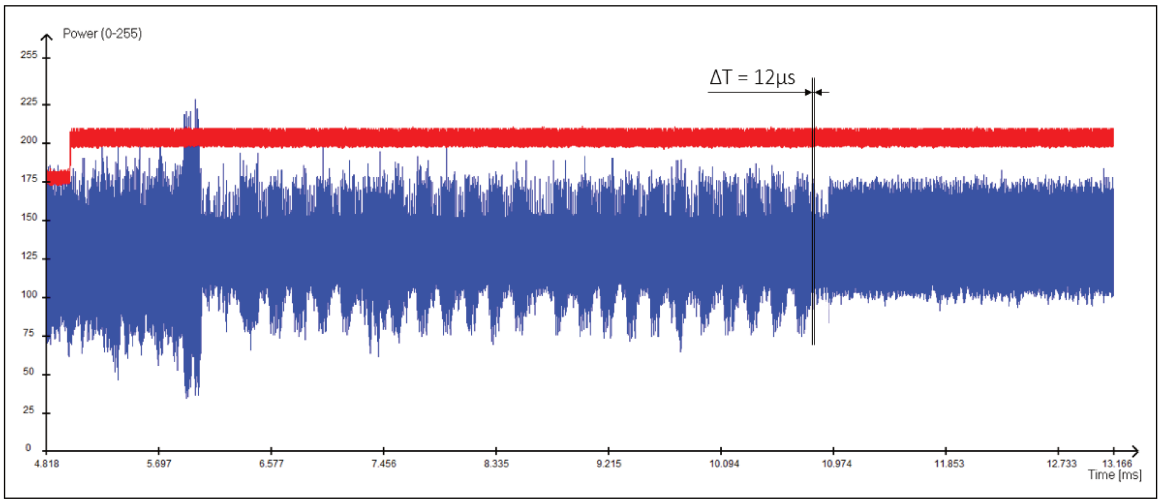


FIGURE 4.32 – Balayage temporel sur la fonction `memcpy()`

Résultats expérimentaux

Sur les 96000 essais de perturbation EM, seuls 214 `<mute>` ont été observés. Sur l'ensemble des données de 256 octets retournés, 13 d'entre elles avaient des octets ne correspondant pas à la valeur de référence retournée pour le *Persistent Storage* ayant l'ID de `0x80`. Le tableau 4.5 liste ces valeurs.

Plusieurs constatations sont faites à l'issue de cette manipulation. Les modifications ne concernent que deux valeurs d'octets. Ainsi on observe des modifications de `0x80` en `0x00` ou `0xFF`. Dans certaines données retournées, les modifications se répètent sur des mots de 32 octets. Il est difficile d'expliquer ce dernier comportement. Comment une unique impulsion de $6ns$ peut-elle modifier les 256 octets d'un tampon mémoire ? Est-ce que les fonctions, API TEE ou autre ont retourné des statuts d'erreurs ? Le fait de ne lire que les 256 premiers octets limite la visibilité dans l'analyse des erreurs et nous passons certainement à coté d'informations plus importantes. Afin de confirmer cette éventualité, notre procédure et nos outils de test ont été améliorés.

4.3.5.3.2 Amélioration de la méthode et des outils

La modification du logiciel de gestion des manipulations pour lire 4096 octets représente un travail conséquent. De plus, la durée de transfert pour 4096 octets est plus importante que celle pour 256 octets. C'est pourquoi, dans un premier temps, des modifications dans la manière d'observer les données retournées ont été effectuées. Les informations souhaitées

Pas temporel	Valeur de retournées (256 octets)
Réf	80 80 80 80 80 80 80 80 80 ··· 80 80 80 80 80 80 80
000872	80 80 80 80 FF 80 80 80 80 ··· 80 80 80 80 80 80 80
001004	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF
002683	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF
002776	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF
003270	80 80 80 80 80 80 80 80 80 ··· 80 80 80 FF 80 80 80
005654	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF
006240	80 80 80 80 80 80 80 80 80 ··· 80 80 FF 80 80 80 80
007743	80 80 80 80 80 80 80 80 80 ··· 80 80 80 00 00 00 FF
008705	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF
010143	FF 80 00 80 FF 80 00 80 FF ··· 00 80 FF 80 00 80 FF
010456	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF
010780	80 80 80 80 80 80 80 80 FF ··· 80 80 80 80 80 80 80
011360	FF 00 00 00 FF 00 00 00 FF ··· 00 00 FF 00 00 00 FF

Tableau 4.5 – Données retournées ne correspondant pas à la valeur de référence lors de la perturbation du `memcpy()`

concernent la présence de valeurs différentes dans le retour de données et l'existence d'un statut d'erreur renvoyé par l'une des fonctions impliquées dans le processus « 3.2 Read Persistent Storage ». Pour cela, les deux modifications suivantes ont été effectuées :

► Renvoi de sommes de contrôle sur les 4096 octets

Afin de détecter des modifications dans les données, ce ne sont plus directement les valeurs des octets du *Persistent Storage* qui sont renvoyées mais des sommes de contrôle (*checksum*) calculées selon l'équation 4.1. Les 4096 octets sont découpés en 16 paquets de 256 octets et les *checksums* additionnent ces derniers dans chaque paquet. Ainsi ce sont 16 cœfficients $CS_{0\leq i\leq 15}$ de 2 octets qui sont retournés pour chaque appel du processus « 3.2 Read a Persistent Storage ». Pour le *Persistent Storage* n° `0x80`, nous avons pour $\forall i \in [\![0, 15]\!]$, $CS_i =$ `0x8000`.

$$CS_{0\leq i\leq 15} = \sum_{j=0}^{j=255} \mathcal{O}_{(256i+j)} \quad : \quad \text{avec l'octet } \mathcal{O}_n \in \{\mathcal{O}_0, \ldots, \mathcal{O}_{4095}\} \tag{4.1}$$

► Renvoi du statut des fonctions de l'API du TEE

Le statut des fonctions de l'API du TEE est codé sur 4 octets. Il s'agit d'une valeur donnant le bilan du déroulement des opérations. Pour un fonctionnement normal c'est la valeur `'TEE_SUCCESS = 0x0000_0000'` qui est retournée. Le processus « 3.2 Read a Persistent Storage » comporte deux fonctions du TEE qui retournent un statut : `TEE_OpenPersistentObject()` et `TEE_ReadObjectData()`. Ces statuts sont ajoutés dans

le renvoi des données. Avec ces modifications, à chaque appel du processus « 3.2 Read a Persistent Storage » ce sont 40 octets qui sont retournés :

$$\texttt{Statut\_Fct1}, \texttt{Statut\_Fct2}, CS_{0 \leq i \leq 15}$$

4.3.5.3.3 Perturbation de la fonction `TEE_ReadObjectData()`

La fonction ciblée dans cette partie est la fonction `TEE_ReadtObjectData()` du processus « 3.2 Read a Persistent Storage ». Elle est issue des API du TEE. Aucune information à son sujet n'étant à notre disposition, nous travaillons en « boîte noire ». Cependant, il s'agit d'une étape essentielle dans le processus de *Secure Storage*, c'est pourquoi nous supposons que s'il existe une vulnérabilité dans le code, une perturbation peut générer le renvoi de données différentes.

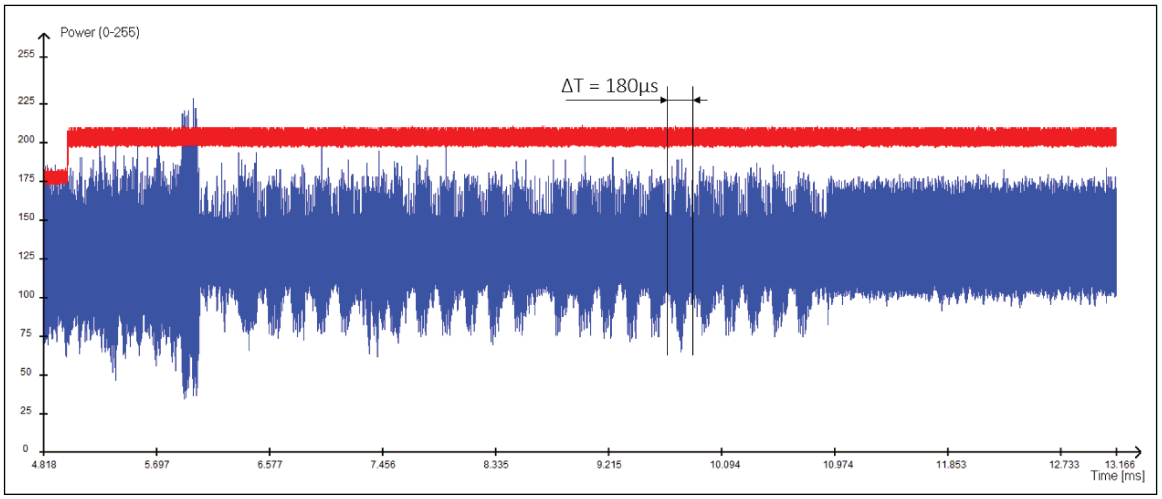


FIGURE 4.33 – Balayage temporel sur la fonction `TEE_ReadtObjectData()`

Procédure d'injection EM.

L'analyse *side-channel* de la Section 4.3.4.3 a permis de localiser temporellement l'exécution de la fonction `TEE_ReadtObjectData()`. Un balayage temporel d'injections EM est effectué sur l'ensemble de son exécution. Cela représente une fenêtre temporelle de $\Delta T = 180\mu s$ (voir Fig.4.33). Plusieurs campagnes d'injections EM ont été effectuées en considérant différentes configurations de pas, de durée et d'amplitude d'impulsion A_{inj} et ΔT_{inj}. Nous présentons dans cette étude une synthèse des résultats les plus significatifs.

Synthèse des résultats expérimentaux

Deux types de comportements sont identifiés. Une modification des statuts retournés par les fonctions impliquées dans le processus « 3.2 Read a Persistent Storage » et la modification des valeurs dans les sommes de contrôles. Ce dernier comportement est observé uniquement lorsque il n'y a pas de statut d'erreur, autrement aucune donnée n'est retournée par le processus.

► **Statut d'erreur des fonctions**

Codes d'erreur retournés par la fonction `TEE_OpenPersistentObject()`
`0xF010_0001: TEE_ERROR_CORRUPT_OBJECT`
`0xFFFF_0006: TEE_ERROR_BAD_PARAMETERS`
`0xFFFF_0008: TEE_ERROR_ITEM_NOT_FOUND`
Codes d'erreur retourné par la fonction `TEE_ReadtObjectData()`
`0xF010_0001: TEE_ERROR_CORRUPT_OBJECT`
`0xFFFF_0006: TEE_ERROR_BAD_PARAMETERS`
`0x5200_F600: Erreur non-explicite.`

Tableau 4.6 – Synthèse des codes d'erreur retournés lors de la perturbation de la fonction `TEE_ReadtObjectData()` avec une unique TA.

Le Tableau 4.6 liste les différents codes d'erreur retournés par chacune des fonctions. Nous donnons les suppositions quant à leurs causes :

- `TEE_ERROR_CORRUPT_OBJECT`.

  Ce code d'erreur est retourné par les deux fonctions malgré le fait que les injections EM soient temporellement localisées sur la fonction `TEE_ReadtObjectData()`. Cela peut s'expliquer en considérant qu'une modification des données du *Persistent Storage* est effective à l'issue d'une impulsion EM mais que la fonction `TEE_OpenPersistentObject()` ne le détecte qu'au court de l'appel suivant. Ceci laisse supposer qu'une vérification de l'intégrité des données est effectuée dans chacune des deux fonctions. Plus de rétroconception est requise pour éclaircir ce point et définir comment cette vérification d'intégrité est faite.

- `TEE_ERROR_BAD_PARAMETERS`.

  Ce code d'erreur est intéressant du point de vue de notre attaque. Il indique que l'un des arguments passés à la fonction `TEE_ReadtObjectData()` a été modifié en une valeur incorrecte. En revanche, nous n'expliquons pas le fait que ce code soit retourné par les deux fonctions.

- `TEE_ERROR_ITEM_NOT_FOUND`.

  Ce code est renvoyé par la fonction `TEE_OpenPersistentObject()` lorsque l'ID passé en argument de la fonction n'existe pas. Ceci confirme l'observation faite sur le code

d'erreur précédent : l'ID = `0x80` a été modifié en une valeur qui n'est pas dans la liste des ID créés. Nous n'expliquons pas non plus, le fait que ce soit la fonction `TEE_OpenPersistentObject()` et non pas `TEE_ReadtObjectData()` qui retourne cette erreur.

- `Erreur non-explicite`.

  Lors d'une perturbation EM, la fonction `TEE_ReadtObjectData()` a retourné un code d'erreur qui n'est pas clairement défini dans la documentation destinée aux développeurs du TEE. S'agit-il d'un code indiquant une combinaison d'erreurs par effet de masque ou l'effet d'une perturbation qui a modifié le statut retourné par la fonction ciblée ? Dans tout les cas, nous n'avons pas été en mesure de comprendre ce qui avait été impacté avec la documentation dont nous disposions.

L'analyse des codes d'erreur montre la capacité de modifier les arguments des fonctions impliquées dans le processus « 3.2 Read a Persistent Storage ». Cependant, plus d'investigations sont requises pour comprendre le fait que nous ciblions la fonction `TEE_ReadtObjectData()` et que certaines erreurs ne sont renvoyées que par `TEE_OpenPersistentObject()`. D'autre part, il serait intéressant d'étudier ce qu'il adviendrait si nous réussissions à modifier l'ID = `0x80` en un autre ID existant. Est-ce que la fonction `TEE_ReadtObjectData()` retournerait les données de ce *Persitent Storage* ?

► **checksum différents**

	Valeur des 16 *checksum* calculés sur 4096 octets (32 octets)
(Réf.)	80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00
	80 00 70 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00
	80 00 80 00 80 00 7F 11 4E 8E 7E EE 80 D6 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00
	38 0C 00 00 15 FA 0E 28 29 26 00 00 18 E3 1F AF 1B 5F 09 44 19 40 34 E8 0D BC 11 2D 12 E1 53 DF
	80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 6F F0 74 00 73 F8 79 FA 7D FE 74 00 63 F8 74 00
	3A A5 00 00 15 FA 0E 28 2A D9 00 00 18 E3 1F AF 1C 2C 09 44 19 40 34 E8 0D A3 11 2D 12 E1 53 DF
	80 00 80 00 80 00 80 00 80 00 74 00 7C 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00
	80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 67 F8 80 00 80 00 77 F8 67 F8 77 F8
	80 00 80 00 80 00 80 00 80 00 80 00 7F F2 7F FE 80 00 7F F8 78 00 80 00 7F FA 7F F6 77 FA 7F F6
	7A FA 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00
	80 00 80 00 80 00 80 00 77 F8 6B F8 65 FA 7D FE 77 F8 70 00 6F F0 80 00 80 00 80 00 5B F8 75 FA
	80 00 74 00 50 00 7C 00 77 F8 70 00 70 00 77 F8 77 F8 74 00 6B F0 77 F8 79 FA 7D FE 60 00 80 00
	[...]

Tableau 4.7 – *checksum* sur les 4096 octets ne correspondant pas à la valeur de référence lors de la perturbation de l'API `TEE_ReadObjectData()`

Le Tableau 4.7 donne un extrait des valeurs de *checksum* calculées sur le *Persistent Storage* avec l'ID = `0x80` lorsqu'une modification a été observée. Ces informations confirment que plus des 256 premiers octets sont modifiés et que la première manipulation n'était pas en

mesure d'observer ce phénomène. De plus, on observe que la taille des données modifiées concerne tantôt une unique valeur de *Checksum* = `0x8000`, tantôt tous les *checksums*, comme si la modification avait eu lieu sur des valeurs réparties sur l'ensemble des 4096 octets. La valeur des données est nécessaire pour analyser le comportement de la fonction perturbée. Nous allons donc modifier le logiciel du PC de contrôle du banc EM pour qu'il renvoie les 4096 octets de données. Cependant, au lieu de poursuivre des expérimentations sur une unique TA, forts de nos améliorations d'observations, nous allons directement étudier la perturbation de la fonction `TEE_ReadtObjectData()` avec la présence de deux TA dans le même container.

4.3.5.4 Expérimentations avec deux TA

A ce stade de nos expérimentations, la TA perturbée partage son contexte d'exécution avec une autre. Les deux TA ont le même code source, appartiennent au même container, et ont pour seule différence leur nom et leur numéro d'identification.

4.3.5.4.1 Configuration des TA

La *Trusted Application* nommée TA 1 est identique à celle utilisée dans les expérimentations sur une unique TA, Section 4.3.5.3. La seconde *Trusted Application* nommée TA 2 utilise le même code mais ne contient qu'un unique *Persistent Storage* avec une ID = `0x80`. Afin de différencier les données contenues dans chaque application, la TA 2 contient 4096 octets avec la répétition de la valeur `0xFABE` (voir Fig.4.34).

TA 1
256 *Persistent Storage*

ID	données (4096 octets)
`0x00`	`0x00 0x00 ··· 0x00 0x00`
⋮	⋮
`0x80`	`0x80 0x80 ··· 0x80 0x80`
⋮	⋮
`0xFF`	`0xFF 0xFF ··· 0xFF 0xFF`

TA 2
1 *Persistent Storage*

ID	données (4096 octets)
`0x80`	`0xFA 0xBE ··· 0xFA 0xBE`

FIGURE 4.34 – *Persistent Storages* créés dans chaque TA pour nos expérimentations. Les deux TA sont dans le même container de TEE.

4.3.5.4.2 Amélioration des outils

Comme la partie des expérimentations sur une seule TA le montre, l'amélioration de nos outils est nécessaire afin d'obtenir un maximum d'informations pour comprendre les résultats observés. Dorénavant, le PC de contrôle du banc EM disposera des deux informations suivantes :

- Les statuts des fonctions de l'API du TEE.

- Les valeurs des 4096 octets renvoyés.

Ces améliorations nous permettent de détecter n'importe quelle modification à l'issue d'une perturbation EM. En revanche, la lecture de 4096 octets ralentit le déroulement des séquences de tests.

4.3.5.4.3 Validation des modifications d'outils sur la TA 1

Cette étape a pour objectif d'observer si la présence d'une seconde TA modifie le type des perturbations susceptibles d'être générées par rapport aux perturbations obtenues avec une unique TA dans la Section 4.3.5.3. Dans cette partie, nous sollicitons la TA 1.

Procédure d'injection EM

La routine de test consiste à appeler le processus « 3.2. Read a Persistent Storage » de la TA 1 en demandant à lire les données du *Persistent Storage* ayant l'ID `0x80`. Les perturbations EM sont générées de la même manière que dans l'expérimentation précédente : au-dessus du point sensible mis en avant dans la Section 4.3.5.2 avec une amplitude et une durée de $A_{inj} = +321V$ et $\Delta T_{inj} = 6ns$. Un balayage temporel encadrant l'exécution de la fonction `TEE_ReadtObjectData()` est effectué sur une fenêtre de $\Delta T = 180\mu s$ (voir Fig.4.33). Plusieurs manipulations ont été effectuées afin de définir ces paramètres. Nous ne présentons que les résultats les plus significatifs.

Synthèse des résultats expérimentaux

Les deux types de modifications observées sur l'ensemble des expérimentations concernent les statuts retournés par les fonctions du TEE et les valeurs des données retournées. Ici aussi, ces types de modifications arrivent de manière dissociée. Lorsque les statuts des fonctions indiquent une erreur, aucune donnée n'est retournée.

► **Statuts d'erreur des fonctions**

Codes d'erreur retournés par la fonction `TEE_OpenPersistentObject()`
`0xF010_0001: TEE_ERROR_CORRUPT_OBJECT`
`0xFFFF_0003: TEE_ERROR_ACCESS_CONFLICT`
`0xFFFF_0006: TEE_ERROR_BAD_PARAMETERS`
`0xFFFF_000C: TEE_ERROR_OUT_OFF_MEMORY`
Codes d'erreur retourné par la fonction `TEE_ReadtObjectData()`
`0xFFFF_0006: TEE_ERROR_BAD_PARAMETERS`

Tableau 4.8 – Synthèse des codes d'erreur retournés lors de la perturbation de la fonction `TEE_ReadtObjectData()` de la TA 1

La liste des codes d'erreur retournés par les fonctions est donnée dans le Tableau 4.8. Une partie des ces statuts ont déja été rencontrés et détaillés dans l'expérimentation avec une seule TA. Cependant, deux d'entre eux sont nouveaux :

- `TEE_ERROR_ACCESS_CONFLICT`.

  Ce code d'erreur apparaît lorsque qu'un objet ouvert par la fonction `TEE_OpenPersistentObject()` transgresse des règles établies à son sujet. L'objet en question est la TA 1. Nous supposons que ce code d'erreur se produit car une tentative de lecture non autorisée a été effectuée sans plus de détails.

- `TEE_ERROR_OUT_OF_MEMORY`.

  Pour arriver à ses fins, la fonction `TEE_OpenPersistentObject()` s'appuie sur deux éléments : un tas (*heap*) et un espace mémoire tampon. Si un de ces éléments est plein, le code d'erreur `TEE_ERROR_OUT_OF_MEMORY` est retourné. Du point de vue de notre expérimentation, ce statut est intéressant car il indique que, durant sa phase de lecture, un des éléments mémoire est excédé. Ceci a pu se produire si la perturbation impacte la copie de données vers le tampon mémoire. Un surplus de données cause un *memory dump* qui en l'occurrence a été détecté.

► Données retournées différentes

Les données renvoyées par le *Persistent Storage* d'ID `0x80` comportaient deux types de modifications. La première est la mise à la valeur `'00'` de portions de données. La deuxième est l'apparition de données de valeurs différentes. Dans ce dernier cas, les données différentes consistaient en la répétition d'une valeur de 4 octets se finissant par la valeur `0xFF`. Certains des ces octets sont constants tandis que d'autres fluctuaient d'un bit. L'origine de cette modification n'est pas évidente. Ces modifications sont résumées ci-dessous.

Valeur de référence (4096 octets) :

80 80 80 80 80 80 80 80 80 80 80 80 80 80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 80 80

Mise à `'00'` de portions de données :

80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 [64 bits mis à '0'] 80 80 80 80 80 80 80 80 [...] 80

80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 [64 octets mis à '00'] 80 80 80 80 80 80 80 80 [...] 80

Données différentes parmi les `'80'` :

80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 A4 91 E0 FF A4 91 E0 FF A4 91 E0 FF A4 91 E0 FF
A4 91 E0 FF A4 91 E0 FF A4 91 E0 FF A4 91 E0 FF 80 80 80 80 80 80 80 80 80 80 80 80 [...] 80

80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 18 13 34 FF 18 14 34 FF 18 14 35 FF 19 16 35 FF
19 16 36 FF 19 14 35 FF 18 14 36 FF 16 13 33 FF 80 80 80 80 80 80 80 80 80 80 80 80 [...] 80

80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 16 13 34 FF 15 12 33 FF 14 12 32 FF 15 12 32 FF
15 12 32 FF 15 13 33 FF 15 12 33 FF 14 11 30 FF 80 80 80 80 80 80 80 80 80 80 80 80 [...] 80

80 [...] 80 80 80 80 80 80 80 80 80 80 80 80 80 A6 95 E5 FF A6 95 E5 FF A6 95 E5 FF A6 95 E5 FF
A6 95 E5 FF A6 95 E5 FF A6 95 E5 FF A6 95 E5 FF 80 80 80 80 80 80 80 80 80 80 80 80 [...] 80

Les informations apportées par l'amélioration de nos outils nous permettent de constater qu'aucune donnée sensible provenant d'un autre emplacement mémoire n'a été renvoyée.

4.3.5.4.4 Perturbation de la fonction `TEE_ReadObjectData()` avec deux TA

Cette expérimentation utilise les deux *Trusted Applications* TA 1 et TA 2. L'application TA 1 contient 256 *Persistent Storages* de 4096 octets tandis que l'application TA 2 n'en contient qu'un seul. Dans cette partie, nous sollicitons la TA 2 afin d'observer si l'isolation de deux TA dans le même contexte est efficace.

Procédure d'injection EM

La routine de test consiste à appeler le processus « 3.2. Read a Persistent Storage » de la TA 2 en demandant à lire les données du *Persistent Storage* ayant l'ID `0x80`. Les perturbations EM avec une amplitude de $A_{inj} = +321V$ et une durée de $\Delta T_{inj} = 6ns$. La fenêtre temporelle considérée est la même (voir Fig.4.33). Plusieurs manipulations ont été effectuées, seuls les résultats les plus caractéristiques sont donnés.

Synthèse des résultats expérimentaux

Toujours deux types de comportements sont observés : soit un code d'erreur retourné par une des fonctions soit une modification dans les données.

▶ **Statuts d'erreurs des fonctions**

Codes d'erreur retournés par la fonction `TEE_OpenPersistentObject()`
`0xF010_0001: TEE_ERROR_CORRUPT_OBJECT`
`0xFFFF_0006: TEE_ERROR_BAD_PARAMETERS`
`0xFFFF_0008: TEE_ERROR_ITEM_NOT_FOUND`
`0xFFFF_000C: TEE_ERROR_OUT_OFF_MEMORY`
`0x0001_7F7C: Erreur non-explicite.`
`0x00FF_C685: Erreur non-explicite.`
Codes d'erreur retournés par la fonction `TEE_ReadtObjectData()`
`0xFFFF_0006: TEE_ERROR_BAD_PARAMETERS`

Tableau 4.9 – Synthèse des codes d'erreur retournés lors de la perturbation de la fonction `TEE_ReadtObjectData()` de la TA 2

Le Tableau 4.9 liste les codes d'erreur retournés. Aucun nouveau code ne s'est présenté et, comme pour les autres manipulations, la majorité des codes d'erreur est renvoyée par la fonction `TEE_OpenPersistentObject()` alors que la perturbation est générée durant l'exécution de la fonction `TEE_ReadtObjectData()`.

► **Données retournées différentes**

La valeur de référence retournée par le processus « 3.2 Read Persistent Storage » lorsque l'on demande les données contenues dans le *Persistent Storage* ayant l'ID = `0x80` de la TA 2 est composée de la répétition de la valeur `0xFABE` sur 4096 octets. Différents types de modifications de données ont été observés. La majorité d'entre elles ne sont pas exploitables mais montrent la possibilité de modifier la valeur retournée.

Valeur de référence (4096 octets) :
`FA BE FA BE FA BE FA BE FA BE FA BE FA BE [...] FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE`

Comme dans les manipulations précédentes, des portions de données sont mises à la valeur `'00'`. Il a même été observé que les 4096 octets aient la valeur `0x00`. Sans informations plus précises sur le code source, il est difficile d'expliquer ce comportement.

Mise à `'00'` de portions de données :
`FA BE [...] FA BE` [64 octets mis à `'00'`] `FA BE [...] FA BE FA BE` [64 octets mis à `'00'`] `FA BE [...] FA BE`

`FA BE [...] FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE` [3072 octets mis à `'00'`]

Mise à `'00'` de toutes les données :
`00 00 [...] 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`

Des modifications ont également été remarquées dans les valeurs de données renvoyées. Deux comportements ont été observés, le changement de la valeur d'un mot de 4 octets ou la modification d'un large bloc de données. Le premier type de modification fait apparaître un mot de 4 octets commençant par la valeur `'58'` et se terminant par la valeur `'00'`. Le deuxième type de modification concerne un bloc de près de 640 octets. Dans ce block les valeurs `0xBE` prennent tantôt la valeur `0xFF`, tantôt la valeur `0x00`. Ces modifications sont données ci-dessous.

Modification de données :
`FA BE [...] FA BE FA BE FA BE FA BE FA BE 58 F8 00 00 FA BE FA BE FA BE FA BE FA BE [...] FA BE`

`FA BE [...] FA BE FA BE FA BE FA BE FA BE 58 FC 00 00 FA BE FA BE FA BE FA BE FA BE [...] FA BE`

`FA BE [...] FA BE FA BE FA BE FA BE FA BE 58 00 01 00 FA BE FA BE FA BE FA BE FA BE [...] FA BE`

`FA BE [...] FA BE FA BE FA BE FA BE FA BE 58 04 01 00 FA BE FA BE FA BE FA BE FA BE [...] FA BE`

Gros bloc de données modifié :
`FA BE FA BE FA BE FA BE FA BE FA BE FA [...] FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA`
`00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF`
`FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA`
`FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00`
`FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA`
`00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF`

```
FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA
FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00
FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA
00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF
FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA
FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00
FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA
00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF
FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA
FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00
FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA
00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF
FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA
FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA FF FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00
FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA
00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA 00 FA BE FA BE FA BE FA BE FA BE FA BE
FA BE FA BE FA BE FA BE FA BE FA BE [...] FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE
```

Enfin le dernier comportement observé, le plus intéressant pour notre étude, est la présence de données provenant d'une autre partie de la mémoire. En effet, au sein des 4096 octets il a été retourné près de 700 octets totalement différents d'un *bitset*, *bitreset* ou *bitflip* sur les valeurs `0xFABE` :

Renvoi d'autres données :

```
FA BE FA BE FA BE FA BE FA BE FA BE FA [...] FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA
73 AB 63 28 96 A0 F9 16 56 DF 65 F2 AC AD 23 2B D1 D3 5C D8 1C FA 83 36 DD 74 D2 20 6B 0A 46
4F A0 1B 76 0C 2F A8 27 DA 68 A5 45 B2 CD 2C D5 84 B4 07 BB 55 D2 AC 1E 41 46 A7 A7 74 95 0B
BF 21 90 D4 44 0B 38 98 5B 97 38 AA AF 92 81 69 7D CB 3E 92 9B E6 FE 79 13 BC 42 F2 40 A4 D2
CE 2D B3 05 76 02 05 1A 98 5F 71 51 27 A5 C3 8E 5D 19 C3 B6 FD 5E 9F 01 00 01 00 0B 00 00 00
00 00 00 00 24 43 46 47 43 52 45 00 07 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FE FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FE 30 21
30 09 06 05 2B 0E 03 02 1A 05 00 04 14 30 2D 30 0D 06 09 60 86 48 01 65 03 04 02 04 05 00 04
1C 30 31 30 0D 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 30 41 30 0D 06 09 60 86 48 01 65
03 04 02 02 05 00 04 30 30 51 30 0D 06 09 60 86 48 01 65 03 04 02 03 05 00 04 40 00 02 00 00
00 03 00 00 00 05 00 00 00 07 00 00 00 0B 00 00 00 0D 00 00 00 11 00 00 00 98 2F 8A 42 91 44
37 71 CF FB C0 B5 A5 DB B5 E9 5B C2 56 39 F1 11 F1 59 A4 82 3F 92 D5 5E 1C AB 98 AA 07 D8 01
5B 83 12 BE 85 31 24 C3 7D 0C 55 74 5D BE 72 FE B1 DE 80 A7 06 DC 9B 74 F1 9B C1 C1 69 9B E4
86 47 BE EF C6 9D C1 0F CC A1 0C 24 6F 2C E9 2D AA 84 74 4A DC A9 B0 5C DA 88 F9 76 52 51 3E
98 6D C6 31 A8 C8 27 03 B0 C7 7F 59 BF F3 0B E0 C6 47 91 A7 D5 51 63 CA 06 67 29 29 14 85 0A
B7 27 38 21 1B 2E FC 6D 2C 4D 13 0D 38 53 54 73 0A 65 BB 0A 6A 76 2E C9 C2 81 85 2C 72 92 A1
E8 BF A2 4B 66 1A A8 70 8B 4B C2 A3 51 6C C7 19 E8 92 D1 24 06 99 D6 85 35 0E F4 70 A0 6A 10
16 C1 A4 19 08 6C 37 1E 4C 77 48 27 B5 BC B0 34 B3 0C 1C 39 4A AA D8 4E 4F CA 9C 5B F3 6F 2E
68 EE 82 8F 74 6F 63 A5 78 14 78 C8 84 08 02 C7 8C FA FF BE 90 EB 6C 50 A4 F7 A3 F9 BE F2 78
71 C6 22 AE 28 D7 98 2F 8A 42 CD 65 EF 23 91 44 37 71 2F 3B 4D EC CF FB C0 B5 BC DB 89 81 A5
DB B5 E9 38 B5 48 F3 5B C2 56 39 19 D0 05 B6 F1 11 F1 59 9B 4F 19 AF A4 82 3F 92 18 81 6D DA
```

```
D5 5E 1C AB 42 02 03 A3 98 AA 07 D8 BE 6F 70 45 01 5B 83 12 8C B2 E4 4E BE 85 31 24 E2 B4 FF
D5 C3 7D 0C 55 6F 89 7B F2 74 5D BE 72 B1 96 16 3B FE B1 DE 80 35 12 C7 25 A7 06 DC 9B 94 26
69 CF 74 F1 9B C1 D2 4A F1 9E C1 69 9B E4 E3 25 4F 38 86 47 BE EF FA BE FA BE FA BE FA BE FA
BE FA BE FA BE FA BE FA BE [...] BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE FA BE
```

L'explication de la présence de telles valeurs d'octets dans les données renvoyées n'est pas aisé. Une analyse de celle-ci peut aider à en comprendre l'origine. Deux traitements simples apportent rapidement des informations. Il s'agit de la recherche de chaînes de caractères et le désassemblage de code. Le premier traitement consiste à convertir les valeurs hexadécimales des octets en caractères d'alphabets humains. De cette manière il est possible d'identifier, dans ces données, s'il y a des informations intelligibles telles que des noms de variables ou de fonctions, des valeurs, des adresses, etc. Le cas échéant, cela permet de conclure et de localiser la partie de code observée.

FIGURE 4.35 – Conversion des valeurs hexadécimales en caractères UTF-8 imprimables

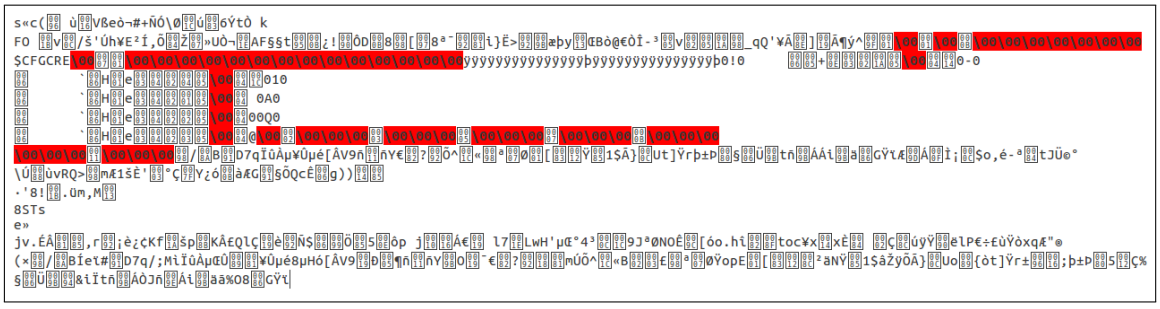


FIGURE 4.36 – Conversion des valeurs hexadécimales en caractères ISO-8859-15 imprimables

Plusieurs systèmes de codage informatique ont été considérés et les résultats de deux d'entre eux sont donnés dans les figures 4.35 et 4.36. Quoi qu'il en soit, aucune information sensée n'a été identifiée avec ce traitement. L'autre traitement consiste à traduire les valeurs hexadécimales en langage d'instructions machine. Avec les informations sur l'architecture du SoC sur lequel est implantée le TEE, nous avons tenté de désassembler ces instructions selon le jeu d'instructions « aarch-64 little-endian ». Ici aussi aucune instructions cohérente n'a été observée. A l'issue des traitements sur ces valeurs d'octets, plusieurs suppositions

ont été formulées.

- Ces données sont-elles chiffrées ?
- Une séquence d'une dizaine de `'00'` suivis d'une trentaine de `'FF'` est présente. S'agit-il de séparateurs ? D'alignements mémoire ?
- Ces données sont-elles pertinentes et sont-elles renvoyées par le TEE ?

Une étude plus approfondie est nécessaire pour déterminer quel genre de données sont retournées à l'issue de la perturbation EM de la fonction `TEE_ReadObjectData()`.

4.3.6 Etude de faisabilité d'attaques d'un TEE : conclusion

Les expérimentations sur un *Trusted Execution Environment* en « boîte noire », ont montré comment une analyse *side-channel* et des perturbations électromagnétiques peuvent être combinées pour attaquer un processus donné. En l'occurrence, le processus ciblé est la lecture d'une valeur sécurisée stockée grâce à la fonctionnalité *Secure Storage* offerte par le TEE testé. La phase d'analyse par canaux cachés est fondamentale pour déterminer certains paramètres de l'étape de perturbation. La phase de perturbation EM a montré deux réactions de ce TEE. La première est une détection d'anomalies et le renvoi de codes d'erreur. Ceci implique que le TEE gère certaines erreurs générées et prend des mesures adéquates. En revanche, ces retours d'erreurs informent sur les processus perturbés et peuvent conduire à l'ajustement des attaques. L'autre type de réaction, plus délicate, est le renvoi de données modifiées sans qu'il y ait de détection de la part du TEE. La majorité des modifications sont de simples *bitset* ou *bitreset* mais peuvent servir d'éventuels chemins d'attaques. Enfin une perturbation a produit le renvoi de données qui n'ont pas pu être identifiées.

La complexité d'un OS TEE, aussi minimaliste soit-il, fait que sans information, sans accès à une documentation claire et à son code source, il n'est pas envisageable de conclure quant à sa robustesse vis-à-vis des attaques matérielles. L'étude en « boîte noire » que nous avons effectuée est soumise à une forte contrainte temporelle. Ceci nous a forcé à cibler une fonction donnée et à essayer de la perturber. Or, sans une analyse plus fine du code source, il n'est pas possible de définir si notre démarche d'attaque est pertinente. Un attaquant tierce, procédera de la même manière mais sans contrainte de temps. Sa persévérance et son opiniâtreté feront que l'éventuelle détection de vulnérabilités aura lieu au détriment du concepteur de TEE.

4.4 Conclusion

Dans ce chapitre, nous avons étudié la faisabilité de la mise en place des attaques matérielles sur trois mécanismes utilisés de manière complémentaire pour isoler l'exécution de processus sensibles au sein d'un SoC. Pour effectuer des expérimentations, le contexte de cette étude nous a amené à considérer des implémentations propriétaires. Bien que leurs principes soient relativement génériques, en appliquant la méthodologie proposée dans le chapitre 2 sur ces implémentations, nous avons souligné leur dépendance vis-à-vis du matériel.

L'analyse des vulnérabilités du *Secure Boot* a montré, dans la Section 4.1.3.1, que celles-ci pouvaient être liées à l'architecture du processeur. Ceci souligne la dépendance du code envers le matériel. La vulnérabilité exploitée dans la Section 4.1.3.2 met en avant le fait qu'un code non protégé est susceptible d'être détourné. Dans les deux cas, une considération d'éventuelles attaques par injections de fautes aurait permis de mettre en place des contre-mesures adaptées afin de limiter la surface d'attaque. Dans le cas présent, l'utilisation d'un DMA et d'une vérification du flot d'exécution auraient compliqué la mise en place des attaques.

L'analyse concernant le TrustZone® a montré l'influence de la technologie utilisée sur l'efficacité de l'attaque. Nous n'avons pas modifié de valeurs susceptibles de casser le contexte sécurisé offert par cette technologie. Cependant, nos expérimentations ont fait ressortir deux aspects remarquables : évaluer les fonctionnalités du TrustZone® indépendamment de la couche logicielle qui l'utilise n'a pas forcément de sens puisque l'efficacité d'une attaque s'appuie sur la prise en compte d'une valeur modifiée ; d'autre part, le travail de [99] prouve que la modification de registre est plus probable dans une phase de transition de valeur. Pour ces raisons, l'évaluation de la fiabilité et l'efficacité du TrustZone® est plus pertinente lorsqu'un logiciel utilise les fonctionnalités d'isolation pour ses processus.

Enfin, l'étude menée sur le *Trusted Execution Environment* souligne la complexité de tels OS et, par conséquent, la difficulté d'évaluer la présence de vulnérabilités exploitables. Nos expérimentations ont souligné le fait qu'une évaluation en « boîte noire » n'est pas suffisante pour caractériser la sécurité d'un produit. Une validation expérimentale après identification des vulnérabilités aurait été mieux adaptée.

L'étude de ces trois mécanismes a démontré la faisabilité d'appliquer des attaques matérielles. Le fait de ne pas considérer ces dernières durant les étapes de conception laissera toujours de potentielles vulnérabilités qui sauront être exploitées. L'ajout de contre-mesures pour protéger des analyses par canaux auxiliaires et des attaques par injections de fautes sont essentielles dans des mécanismes voués à des tâches sécuritaires.

Chapitre 5

Débogage Matériel

Ce chapitre étudie le débogage matériel en tant que menace potentielle pour les systèmes complexes embarqués. En effet, cet outil apporte autant d'avantages aux développeurs que de vulnérabilités aux systèmes. En faisant office de « porte dérobée », il offre un accès privilégié aux divers modules. Par conséquent, un mécanisme de débogage non sécurisé permet de contourner les défenses d'un système, compromettant ainsi la sécurité des données censées être protégées. Aujourd'hui, la communauté des systèmes électroniques a pris conscience de ces risques et propose des solutions afin de protéger le débogage matériel [147, 119, 1]. Cependant, les implémentations matérielles de ces solutions peuvent comporter des vulnérabilités. C'est pourquoi, dans ce chapitre, nous étudions la relation entre le débogage matériel et les attaques matérielles. La Section 5.2 donne un aperçu des protections de sécurité proposées ; puis, dans le contexte de ce travail, la Section 5.3 étudie la faisabilité de contourner ces sécurités au moyen d'attaques matérielles. Ensuite, la Section 5.4 présente une nouvelle exploitation du débogage dans les attaques matérielles, et enfin, une conclusion de ce chapitre est donnée Section 5.5.

Sommaire du chapitre

5.1 Le débogage matériel

En 1990, suite à la publication de la première version de la norme IEEE Std 1149.1 [6], le débogage matériel a massivement été intégré dans les systèmes électroniques. Cet outil élaboré pour la conception, la vérification et la maintenance des circuits, propose des fonctionnalités qui permettent d'accroître considérablement la rapidité de la mise sur le marché des produits. Au fur et à mesure de l'évolution des technologies embarquées, le débogage matériel est devenu un outil indispensable pour leur développement. En parallèle, les pirates informatiques constatant les possibilités offertes par un tel outil, ont rapidement fait émerger la problématique de la cohabitation entre les systèmes de débogage matériel et les données sensibles. En 2002, l'étude [69], a démontré la possibilité d'utiliser le JTAG pour effectuer des accès mémoire de manière frauduleuse. Aujourd'hui, sur les blogs et les forums dédiés à l'informatique, une abondante documentation propose une multitude d'exploitations afin de contourner des systèmes de sécurité. La plupart des manipulations décrites exploitent un accès ouvert au débogage matériel pour extraire ou écrire des portions de code visant à dégrader des fonctionnalités de sécurité.

La majorité des développeurs qui utilisent les outils de débogage du commerce n'a pas besoin de connaître les principes des protocoles bas niveau utilisés. La vision globale du système apporté par les interfaces des outils de débogage leur est suffisante. Dans le contexte de cette étude, nous nous sommes intéressés à ces protocoles, et plus particulièrement à leurs mécanismes de sécurité. Aujourd'hui, les deux protocoles de débogage matériel les plus répandus dans l'électronique sont le JTAG et le SWD pour les cœurs ARM. Les annexes D.1 et D.2 détaillent leurs principes de fonctionnement.

5.1.1 Le JTAG

La norme de débogage matériel JTAG définit un protocole et une architecture qui simplifient le diagnostic et la correction des défaillances dans les systèmes électroniques. Initialement, le *Joint Test Action Group* est le nom du groupe de standardisation qui a rédigé la norme IEEE 1149.1, intitulée « *Standard Test Access Port and Boundary-Scan Architecture* » [6]. Pour plus de simplicité, cette norme est largement désignée par le sigle du groupe de travail. Via un unique connecteur, le port d'accès aux tests (*Test Access Port* : TAP), le JTAG propose trois principales fonctionnalités :

- Le *Boundary-Scan* ou scrutation des frontières en français, est utilisé par les outils de test physique pour tester les courts-circuits et la continuité des pistes d'un appareil électronique. Chaque signal primaire d'entrée et de sortie est complété avec un élément mémoire appelé cellule Boundary-Scan. Ces cellules sont chaînées entre elles pour former le registre Boundary-Scan. Connaissant le schéma électrique du circuit, on applique des signaux sur les broches d'entrée du circuit, puis on relève les signaux des broches de sortie afin de s'assurer de la bonne qualité des pistes du circuit imprimé et des soudures. Pour effectuer ces test, les fabricants des systèmes conformes à la norme IEEE 1149.1, doivent fournir un fichier en langage *Boundary Scan Language Description* (BSDL) qui contient les informations sur la mise en œuvre du *Boundary-Scan* sur les composants compatibles de la carte électronique.

- Le *In-Circuit Debugger* ou *In-Circuit Emulator*, débogage interne au circuit en français, est prévue pour tester les fonctionnalités d'un circuit. En appliquant un ensemble de signaux logiques (appelé vecteur de test) sur les broches d'entrée du circuit, on stimule diverses fonctionnalités, puis on relève les niveaux logiques sur les broches de sortie pour s'assurer qu'ils correspondent aux valeurs attendues.

- Le *Debug Access*, l'accès au débogage en français, est utilisé par les outils de débogage matériel pour accéder aux ressources et aux fonctionnalité internes des puces électroniques. Cette fonctionnalité permet d'accéder aux routines internes des puces rendant accessibles et modifiables leurs ressources et opérations. Elle est utilisée par les outils de débogage logiciel pour vérifier le bon fonctionnement logique des systèmes en accédant par exemple aux registres et aux mémoires. Cette fonctionnalité est également mise en œuvre afin de charger les micrologiciels (*firmware*) dans les systèmes.

Bien que les principes du JTAG soient génériques à tous les types de SoC, les fonctionnalités implémentées sont propres à chaque fabricant, voire à chaque système électronique.

5.1.2 Le SWD

Le *Serial Wire Debug* (SWD) [138] est une solution alternative au JTAG proposée par ARM. Elle fait partie de l'ensemble d'IP spécialisé pour le débogage : ARM CoreSight® [41]. Les éléments de cette infrastructure peuvent être accédés par des outils de débogage externes au système via une interface utilisant seulement deux signaux. A l'instar du JTAG, le SWD permet la programmation et le débogage. Cependant le protocole n'implémente pas de solution pour réaliser du *Boundary-Scan*.

5.1.3 Comparaison des protocoles

Le tableau 5.1 synthétise les principales caractéristiques des deux protocoles et met en avant leurs principales différences.

	JTAG	SWD
Nombre de signaux	4	2
Compatibilité	Tous les composants conformes à la norme IEEE 1149.1	Architectures ARM
Fonctionnalité	Boundary-Scan, débogage, programmation	Débogage programmation
Vitesse maximum	Fixé par le composant le plus lent de la chaîne de scan	Fixé par la vitesse du contrôleur ciblé
Contrôle d'erreur prévu par le protocole	Non	Acquittement (statut) sur 3 bits

Tableau 5.1 – Comparaison entre les protocoles JTAG et SWD

5.2 Les protections des mécanismes de débogage

Ces dernières années, la conjugaison de deux facteurs principaux est responsable de la démocratisation des menaces que représente le débogage matériel. Le premier est la disponibilité d'outils à des prix abordables. En effet, lorsqu'une attaque nécessite du matériel sophistiqué de laboratoire, le risque et le profil des attaquants est plus restreint. Avec des outils de débogage matériel peu onéreux et accessibles à tous, la reproduction d'une attaque se fait à plus grande échelle. Le second facteur responsable est la vulgarisation des méthodes d'attaque sur les blogs et forums d'informatique. La disponibilité de ces informations entraîne une très forte augmentation du nombre de personnes susceptibles d'appliquer les méthodes décrites. L'exemple typique est le pirate qui découvre une vulnérabilité liée au débogage, et qui publie un article sur un blog détaillant la démarche à suivre pour l'exploiter de telle sorte qu'un utilisateur « lambda » puisse reproduire la manipulation, sans forcément en comprendre les subtilités. C'est pourquoi, en parallèle, devant la menace grandissante, la communauté de la sécurité numérique propose des solutions de sécurisation du débogage matériel. Nombre des ces solutions utilisent des mécanismes de sécurité dont la robustesse à été prouvée : cryptographie, certificats, systèmes de *token*, etc. Cependant, les contraintes économiques font que ces solutions sont rarement implémentées, au profit de solutions plus économiques demandant l'ajout de peu de portes logiques et de surface de silicium. Une solution présentant le meilleur compromis entre coût et robustesse sera privilégiée par les industriels.

5.2.1 Solutions ARM CoreSight™

Dans son ensemble de solutions CoreSight™[41], ARM met à disposition des mécanismes de contrôle. Des signaux permettent de gérer l'activation des éléments de débogage en fonction du niveau de privilège de l'état de fonctionnement du système. L'avantage est, qu'avec quelques signaux, toute la chaîne de débogage peut être sécurisée. Ce mécanisme de protection est utilisé pour l'architecture TrustZone® et permet de séparer les différents contextes d'exécution (*Secure* et *NonSecure*). Les sécurités se répartissent selon deux ensembles : les composants de débogage du processeur et ceux du système.

► **Composants de débogage du processeur.** Au vue de la complexité des systèmes sur puce, il est nécessaire que l'ensemble de la communauté comme les concepteurs d'appareils électroniques et les développeurs d'applications, puissent être capables d'effectuer des tests et du débogage, même lorsque l'appareil est sur le terrain et que le logiciel de sécurité a été activé. Pour cela, les outils de débogage matériel sont de puissants alliés, mais représentent également une grande menace lorsqu'ils sont utilisés par les mauvaises personnes. La solution de sécurité CoreSight™ a été conçue pour que l'extension de sécurité TrustZone® puisse proposer une isolation des données, même vis-à-vis des outils de débogage. Pour configurer ses accès, le logiciel *Secure* dispose de quatres signaux. Deux pour les accès au modes privilégiés :

- `SPIDEN` (*Secure Privileged Invasive Debug Enable*), est le signal qui va autoriser les outils de débogage intrusifs comme le SWD ou le JTAG, à accéder aux ressources du processeur lorsqu'il est dans son mode de fonctionnement le plus sensible : *Secure*, privilégié.

- `SPNIDEN` (*Secure Privileged Non-Invasive Debug Enable*), autorise les outils de débogage non intrusifs comme la Trace [19]., à lire les ressources du processeur.

Et deux autres pour les accès au mode utilisateur (*User*) :
- `SUIDEN` (*Secure User mode Invasive Debug Enable*), autorise le débogage intrusif.
- `SUNIDEN` (*Secure User mode Non-Invasive Debug Enable*), autorise le débogage non-intrusif.

De plus, de la même façon que sur les processeurs ARM n'implémentant pas les solutions TrustZone®, il est possible de désactiver l'ensemble des accès au débogage, ce qui inclut les accès au *Normal World*.
- `DBGEN` (*Global invasive Debug Enable*)
- `NIDEN` (*Global non-invasive Debug Enable*)

De cette manière, il est possible d'appliquer une routine de débogage *NonSecure* à un processeur sans qu'elle puisse accéder aux ressources de ce dernier lorsqu'il est en mode *Secure*.

► **Composant de débogage du SoC.** Le standard ARM CoreSight™ permet à des outils externes ou des logiciels de débogage, d'accéder à différents éléments d'un SoC. D'un point de vue débogage CoreSight™, ces éléments peuvent être atteints via l'interface APB. Pour réduire les composants nécessaires, les sécurités TrustZone® des passerelles AXI-to-APB ne sont pas implémentées. Cependant, la problématique de protection des éléments du *Secure World* est toujours de rigueur ; il faut être capable de contrôler les accès même au niveau du débogage. Pour cela, la solution proposée est d'utiliser les mêmes signaux que les processeurs (`SPIDEN` `SPNIDEN` et `DBGEN`) et de les appliquer aux composants de la chaîne CoreSight™. Par exemple, si un outil de débogage externe ou un logiciel de débogage *NonSecure* souhaite mettre un point d'arrêt à une adresse *Secure* lorsque `SPIDEN` n'est pas activé, les modules CoreSight™ n'exécuteront pas la requête. Il faut souligner que, lorsque ce signal est activé, un outil de débogage *NonSecure* peut accéder aux *Secure World*.

Les solutions de sécurité CoreSight™ ne fonctionnent qu'avec des bloc-IP ARM qui implémentent les signaux que nous venons de décrire. Malgré le monopole des composants ARM sur le marché, cette solution ne convient pas pour tous les SoC.

5.2.2 Sécurisation des protocoles de débogage

Les solutions brièvement décrites dans cette section montrent que des études ont proposé de mettre en place des mécanismes qui protègent les protocoles de débogage. Cependant, durant cette étude, aucune d'entre elles n'a été observée comme mécanisme de protection implémenté dans un SoC.

La solution proposée dans le travail [128] protège la confidentialité des données transitant sur la chaîne de scan. Cette solution est préconisée pour les chaînes de débogage

19. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.coresight/index.html

manipulant des secrets comme des clefs cryptographiques ou des mémoires sécurisées. Le principe de cette protection est d'utiliser un algorithme de chiffrement symétrique en entrée et en sortie de chaîne de scan. Seuls les évaluateurs autorisés posséderont une copie de la clef de chiffrement liée au circuit. Grâce à celle-ci, les vecteurs de tests sont chiffrés avant d'être envoyés en entrée de chaîne de scan. Puis avec sa copie de clef secrète, le circuit déchiffre les instructions, procède aux tests, et chiffre à nouveau les données renvoyées sur la sortie de la chaîne de scan. L'évaluateur récupère les données et les déchiffre avec sa copie de clef secrète. Il est préconisé d'utiliser différentes valeurs de clefs par type d'opérations de débogage. Des algorithmes de chiffrements « légers » sont utilisés dans cette solution pour ne pas surcharger les besoins en portes logiques des circuits. Aucune modification du protocole de débogage n'est nécessaire et la confidentialité des données est garantie. Seule la gestion des clefs reste un point sensible.

Une autre solution, proposée dans [65], consiste à fixer la structure de la chaîne de scan uniquement durant le mode de test. Lorsque ce n'est pas le cas, les bascules composant la chaîne sont dynamiquement et aléatoirement assignées à différentes positions. Ces opérations de permutation empêchent d'analyser les données observées sur le signal de sortie, étant donné que le réordonnancement des bascules a lieu à tout moment. Les données sensibles ne sont pas accessibles durant le mode de test. Cette solution fournit un haut niveau de sécurité est intégrable dans plusieurs types de conceptions. Néanmoins le mécanisme de permutation augmente de manière conséquente la surface de silicium requise ainsi que la consommation électrique du circuit.

Dans [119], les auteurs proposent d'utiliser un algorithme de chiffrement par flot pour chiffrer le contenu des échanges JTAG. En effet ces algorithmes nécessitent moins de portes logiques et de puissance électrique que les chiffrements pas blocs. Cependant, une fonction de hachage et un protocole d'authentification des messages sont requis pour procéder à un échange sécurisé des clefs utilisées pour le chiffrement par flot.

5.2.3 Sécurisation de l'accès aux protocoles

Les solutions présentées ici se focalisent sur la protection de l'accès au débogage. Le plus souvent le protocole de débogage reste inchangé, il est simplement encapsulé par une procédure qui le verrouille.

Dans [104], les auteurs décrivent une solution de sécurité basée sur un serveur distribuant des jetons (*token*) selon un protocole sécurisé. Les jetons d'accès sont fournis en fonction des droits alloués à un utilisateur qui se connecte avec un identifiant et un mot de passe à un serveur sécurisé. Ce système propose un contrôle au cas par cas de l'accès au débogage. Toutefois, cette solution nécessite l'ajout de divers modules coûteux en consommation électrique et en surface de silicium (chiffrement par bloc, RNG, mémoire non volatile,...). De plus, les identifiants et jetons nécessitent la gestion de bases de données et de serveurs distants. L'ensemble est relativement lourd à gérer et les auteurs consacrent leurs solutions aux systèmes nécessitant une protection particulière, comme les mobiles implémentant un TEE par exemple.

Les auteurs de [45] proposent une sécurité établie sur du chiffrement asymétrique selon le protocole de Schnorr. Leur schéma de protection assure un échange de clefs sécurisées, certifiés sans révéler d'indice sur le secret. Un algorithme de chiffrement basé sur les courbes elliptiques est préféré pour des raisons de limitation de superficie de silicium nécessaire. Cette solution permet de s'affranchir du problème de gestion des clefs, inhérent aux algorithmes symétriques, et de garantir une sécurité dans les échanges JTAG. En revanche, les auteurs informent qu'ils ne considèrent pas, dans leur solution, les menaces que représentent les attaques *side-channel* sur les clefs privées.

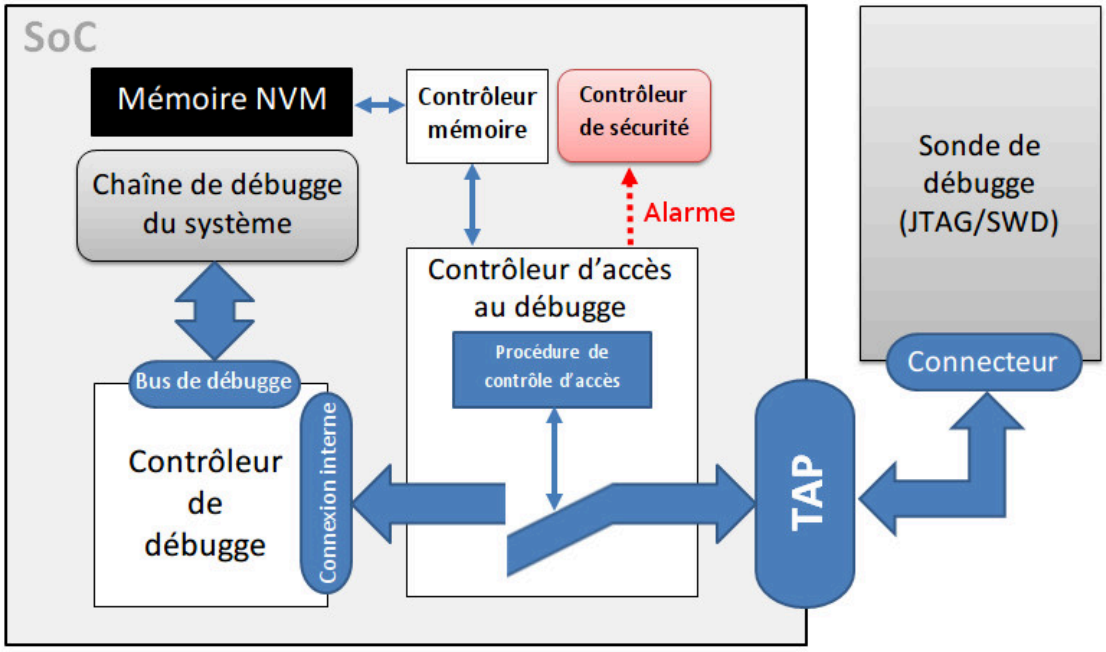


FIGURE 5.1 – Mécanisme de verrouillage de l'accès au débogage basé sur la comparaison avec une valeur stockée dans une mémoire persistante du système.

Les industriels proposent également certaines solutions qui respectent la règle du compromis entre efficacité et coût. Majoritairement, les solutions que nous avons pu observer durant cette étude sont basées sur un mécanisme de comparaison avec une valeur de référence stockée dans une mémoire persistante et sécurisée. La figure 5.1 présente le schéma générique de ces mécanismes. Ce type de solutions est souvent privilégié car il permet de s'affranchir de la gestion de clefs et d'identifiants. Le niveau de sécurité semble acceptable ; en outre le coût et la complexité sont réduits.

C'est le cas, par exemple, de la solution proposée dans [126]. Trois niveaux de sécurité sont disponibles pour accéder au JTAG. Un niveau sans aucune restriction, un autre verrouillé par un mot de passe et un dernier condamnant définitivement l'accès au débogage. La configuration de ces niveaux ainsi que la gestion du mot de passe s'effectuent avec les

OTP (voir Annexe A.3.1.2). Une fois les fusibles programmés, le mode d'accès au débogage est configuré de manière définitive. Lorsque la solution du mot de passe est choisie, la sonde de débogage doit envoyer la même séquence que celle contenue dans les OTP pour déverrouiller le JTAG jusqu'au prochain redémarrage. D'éventuelles alarmes peuvent être déclenchées en cas d'erreur. Un autre mécanisme proposé dans [135] est plus spécifique aux microcontrôleurs embarquant du code sensible dans une flash interne dont le seul moyen d'accès extérieur est le débogage. La solution est basée sur une valeur de référence stockée dans cette flash. L'objectif de ce mécanisme est de protéger l'accès au code sensible. Pour cela, les développeurs chargent leur code grâce aux outils de débogage, puis ils modifient une valeur contenue à une adresse spécifique de la flash. Lorsque cette donnée est différente de sa valeur initiale, l'accès au débogage est verrouillé. Pour déverrouiller l'accès, il est nécessaire d'effacer l'intégralité de la flash afin de réinitialiser toutes les valeurs, et en particulier celle de référence. C'est ce type de mécanisme que nous allons étudier dans la section suivante.

5.3 Etude de faisabilité d'attaque des mécanismes de verrouillage

L'objectif de cette partie est de montrer qu'en appliquant une attaque par injection de faute, il est possible de contourner le verrouillage d'accès au débogage. La mise en place de cette attaque suit la méthodologie du chapitre 2 et chaque étape est détaillée dans les sections correspondantes.

5.3.1 Véhicule de Test

L'étude effectuée dans cette section s'est déroulée dans le cadre d'une évaluation sécuritaire d'un appareil basé sur un système embarqué destiné au grand public. Pour des raisons de confidentialité, nous décrirons les principes de notre démarche sans donner de détails techniques.

Le microcontrôleur considéré dans cette étude est un système intégrant un processeur ARM Cortex-M3 d'architecture 32 bits cadencé jusqu'à $48M$Hz. Une mémoire RAM de 128 ko et une flash de 1024 ko interne au contrôleur, lui permettent de faire fonctionner un micrologiciel propriétaire. Ce système comporte un accélérateur matériel AES qui stocke ses clefs de chiffrement dans une partie de la mémoire flash. Un module MPU est présent pour contrôler les accès aux données dans la mémoire. Le protocole de débogage utilisé par ce système est le SWD. Ce dernier est employé pour la programmation et le chargement des données. Une fois le système configuré, le verrouillage de l'accès au débogage s'effectue en modifiant la valeur du « statut de verrouillage du débogage » contenu dans la flash. Cette valeur définit l'état de verrouillage du SWD et autorise le débogage uniquement lorsque elle est à sa valeur initiale, à savoir `0xFFFF_FFFF`. Ainsi pour déverrouiller l'accès, il est nécessaire de réinitialiser la flash, ce qui efface le contenu des données. De cette manière les concepteurs du système garantissent la confidentialité, l'intégrité et l'authenticité du code et des données.

Ce processeur est implanté sur une carte de développement possédant toutes les caractéristiques pour utiliser ses fonctionnalités.

5.3.2 Analyse des vulnérabilités

Nous allons présenter comment une analyse du fonctionnement permet de révéler d'éventuelles failles exploitables. Les principes de ce fonctionnement, décrits par la suite, sont issus de documentations disponibles légalement et librement sur internet.

5.3.2.1 Principe du verrouillage de l'interface de débogage

Dans le Cortex-M3, le SWD propose une interface compète pour déboguer le système au travers du bus AHB en accédant aux registres du contrôleur AHB-AP, comme spécifié dans [70] et [113]. Une fois le système sur le terrain, pour protéger le code et les données sensibles, le microcontrôleur que nous ciblons dispose d'un mécanisme de verrouillage. Un contrôleur d'accès faisant office de commutateur est placé entre le port d'accès du SWD et le AHB-AP (voir Fig. 5.2). Lorsque le débogage est verrouillé, le SWD n'accède qu'au contrôleur d'accès tandis que lorsque le débogage est déverrouillé, le contrôleur devient « transparent » et le SWD accède directement au AHB-AP.

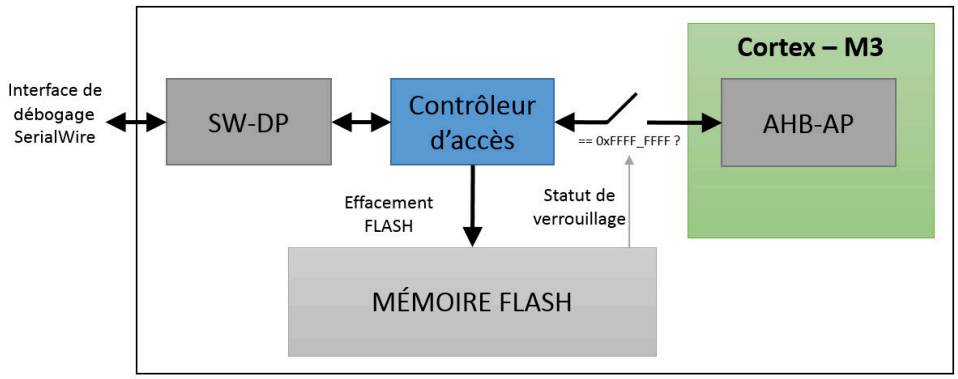


FIGURE 5.2 – Architecture de l'interface de débogage

5.3.2.1.1 Verrouillage du débogage.

Le verrouillage s'effectue en écrivant une valeur différente de `0xFFFF_FFFF` à l'adresse du « statut de verrouillage du débogage » dans la flash. Cette opération est réalisée avec le SWD qui a accès à l'intégralité des ressources de la puce. L'état de verrouillage du système ne sera pris en compte qu'à son prochain démarrage.

5.3.2.1.2 Déverrouillage « normal » du débogage.

Pour déverrouiller l'accès au débogage, un jeu de commandes spécifiques doit être envoyé aux registres du contrôleur d'accès depuis l'interface SWD. La séquence de déverrouillage peut se résumer à ceci :

1. Connecter la sonde au port de débogage du SWD (SWD-DP).
2. Dans le registre d'effacement de flash, mettre la valeur démarrant cette procédure. Attendre l'information d'accomplissement de la tâche (*flag* dans un registre de statut du contrôleur). La valeur par défaut d'une cellule mémoire effacée est `0xFFFF_FFFF`. Le « statut de verrouillage du débogage » se trouvant à la fin d'une page mémoire, l'effacement de la page supprimera d'abord les données avant de déverrouiller l'accès au débogage.
3. Redémarrer le système pour que le contrôleur d'accès prenne en compte la nouvelle valeur du statut de verrouillage.

5.3.2.2 Réflexion sur les potentielles vulnérabilités

En considérant les procédures de verrouillage/déverrouillage du débogage, des points critiques sont remarquables. Ces points représentent les instants auxquels des choix sont faits en comparant des valeurs. A chaque redémarrage du système, la valeur du « statut de verrouillage du débogage » dans la flash est lue afin de définir l'état du SWD. La séquence de démarrage issue de la documentation du fabriquant est donnée figure 5.3. Les deux étapes grisées sont celles qui présentent le meilleur angle d'attaque pour une perturbation électromagnétique :

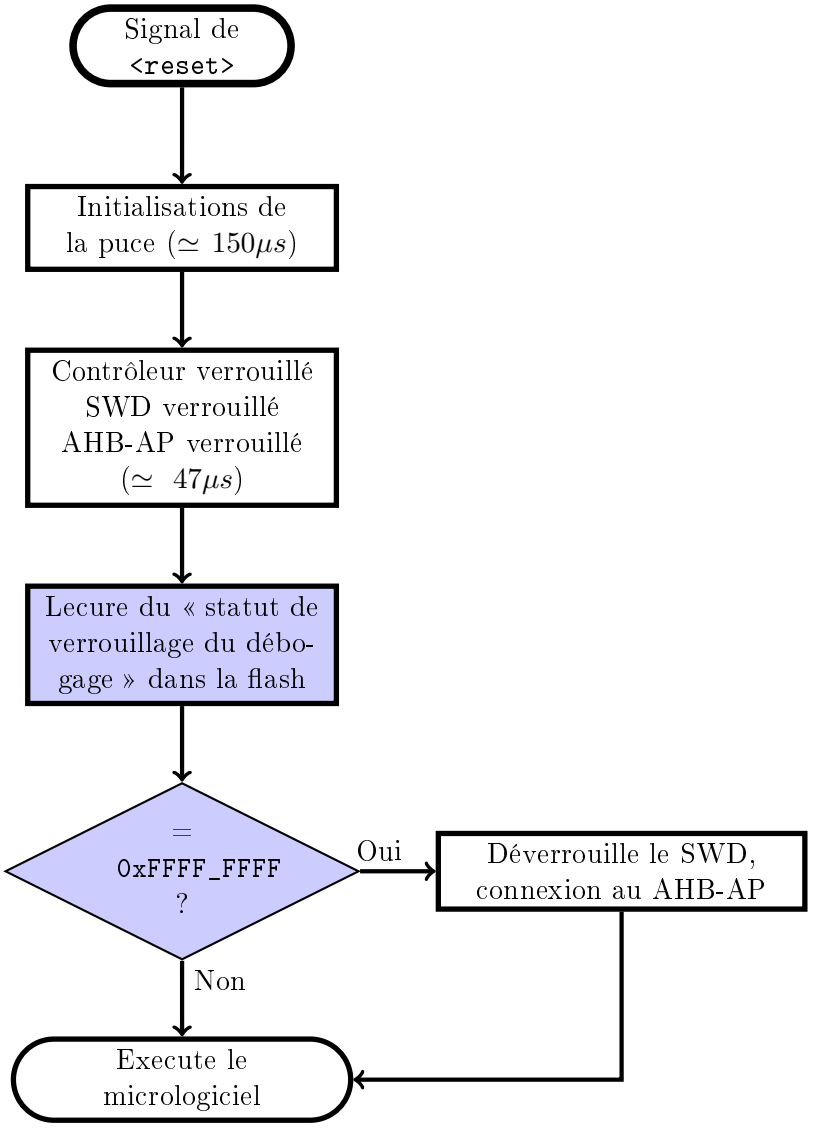


FIGURE 5.3 – Schéma de la séquence de démarrage du microcontrôleur. Les cases grisées représentent les étapes susceptibles de présenter des vulnérabilités face aux perturbations électromagnétiques.

- Lecture du statut de verrouillage dans la flash. Une perturbation durant cette lecture peut amener à lire `0xFFFF_FFFF` au lieu de la valeur réelle.
- La comparaison de la valeur lue avec `0xFFFF_FFFF`. Si la perturbation arrive durant la comparaison, elle est susceptible de produire le résultat inverse de celui prévu initialement par les concepteurs du système.

Dans les deux cas, la réussite de cette attaque est conditionnée par l'absence de toute contre-mesure dans le système, comme une redondance d'opérations par exemple. Le fait de travailler en pseudo-« boîte noire », avec uniquement de la documentation destinée aux développeurs, ne nous permet que de supposer d'éventuels chemins d'attaques. Ce sont les expérimentations et le succès d'une attaque qui nous permettront de conclure.

5.3.2.3 Principe des expérimentations

L'objectif des expérimentations est de perturber électromagnétiquement les opérations déterminées dans la réflexion précédente. Pour cela, une analyse *side-channel* détermine les instants de ces opérations et une campagne d'injection EM évaluera la possibilité de les perturber. Les bancs de tests utilisés sont ceux décrits dans le chapitre 2. Il est nécessaire d'automatiser les requêtes SWD pour pouvoir les intégrer dans le processus d'expérimentation. La gestion de la sonde de débogage SWD ainsi que les redémarrages du système évalué sont effectués par le Raspberry Pi. Un GPIO de ce dernier envoie un signal de `<reset>` à la carte de développement intégrant le système évalué.

5.3.3 Analyse par canaux cachés

L'analyse par canaux cachés a été effectuée avec le banc de mesure EM décrit dans la section 2.4 du chapitre 2. La sonde de mesure utilisée est une LANGER ICR HH-250 avec une bande passante de $6GHz$. L'échantillonnage de l'oscilloscope à la fréquence de $5GS/s$, est déclenché par le signal de `<reset>` du Rasperry Pi. Pour identifier l'instant de lecture du « statut de verrouillage du débogage » dans la flash et l'instant de la comparaison des valeurs, des mesures EM ont été effectuées tantôt lorsque le débogage de la carte de développement est verrouillé, tantôt lorsque il est ouvert. La sonde de mesures a été positionnée manuellement au-dessus d'emplacements émettant des signaux électromagnétiques caractéristiques. Parmi l'ensemble des mesures, deux emplacements reflètent l'activité du processeur. Les Fig. 5.4 et 5.5 montrent les mesures effectuées en ces points.

La séquence de démarrage est localisée dans les 200 premières microsecondes après la montée du signal de `<reset>`. Par conséquent, la décision de verrouillage du débogage est prise durant la période délimitée par les pointillés des deux *Zooms* sur les figures 5.4 et 5.5. Les motifs présents dans la suite des traces représentent l'activité du processeur exécutant le micrologiciel propriétaire une fois le système démarré.

Les *Zooms* 1 et 2 sont respectivement donnés par les figures 5.6 et 5.7. Ces dernières ont la même échelle de temps et présentent plusieurs séquences distinctes :

1. Après la montée du signal de `<reset>`, pendant une durée approximative de $120\mu s$, aucun signal n'est mesuré excepté un pic sur la Fig. 5.7. Ce pic étant présent quel que soit l'état d'accès au débogage et durant la période pendant laquelle les alimentations

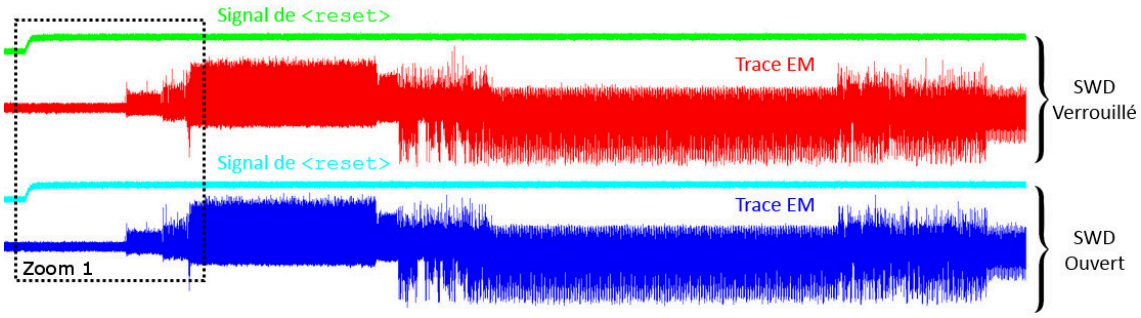


FIGURE 5.4 – Mesures électromagnétiques au-dessus de la position 1

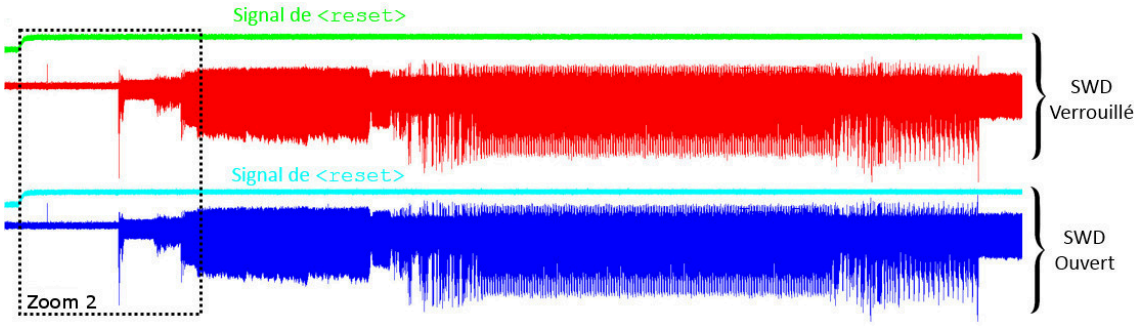


FIGURE 5.5 – Mesures électromagnétiques au-dessus de la position 2

et horloges sont en cours d'initialisation, il est raisonnable de penser qu'il ne s'agit pas d'un processus impliqué dans le mécanisme ciblé.

2. Une seconde période d'approximativement $40\mu s$ comporte un premier processus. Des pics représentant le travail cadencé d'un module sont observables.
3. Une troisième période $\simeq 30\mu s$ comporte également des pics cadencés à la même fréquence que celle des pics observés dans la période précédente, mais avec une amplitude supérieure. Ces pics sont suivis de signaux avec une plus grande amplitude, que nous supposons être liés au début de l'exécution du micrologiciel par le CPU.

Par analogie avec les différentes phases de démarrage décrites par la figure 5.3 :

- Les séquences 1 et 2 constituent les 150 premières microsecondes du démarrage. Il s'agit d'une étape d'initialisation du système après la mise sous tension. Durant cette phase, les tensions d'alimentation et les signaux d'horloge se stabilisent. Les deux figures montrent que la durée indiquée dans la documentation est une estimation de la valeur réelle qui dépend du matériel et des conditions physiques de démarrage (ex. température, alimentation).
- La $3^{\text{ème}}$ séquence représente les $47\mu s$ durant lesquelles le contrôleur d'accès au débogage fonctionne, avant que le CPU prenne le relai. C'est durant cette période que le contrôleur lit la valeur du « statut de verrouillage du débogage » dans la flash et la

compare à `0xFFFF_FFFF`.

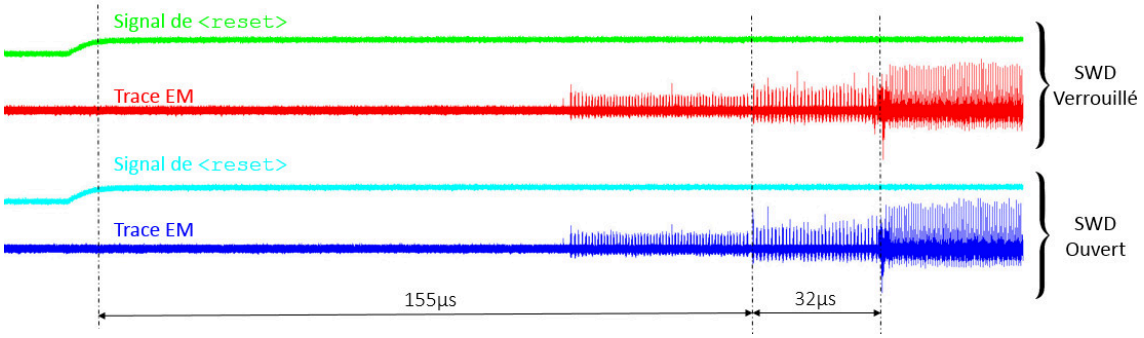


FIGURE 5.6 – Zoom 1 de la Fig.5.4

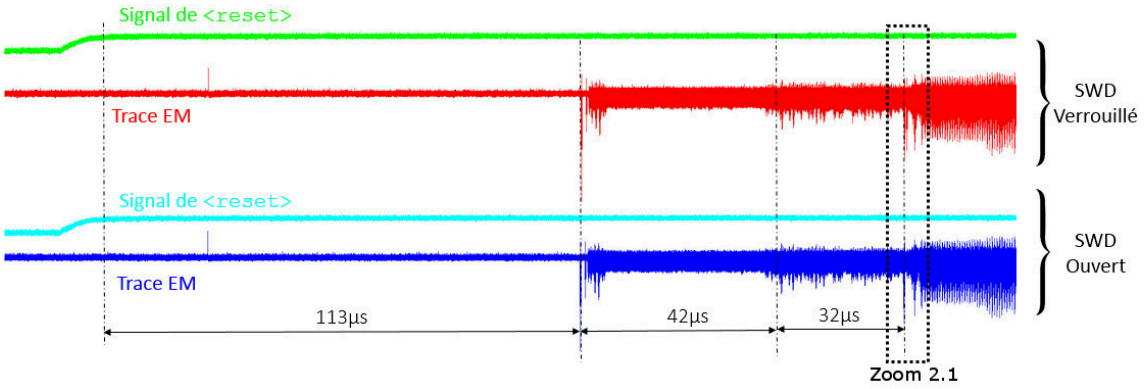


FIGURE 5.7 – Zoom 2 de la Fig.5.5

La correspondance entre la durée de la séquence 3 et celle des $47\mu s$ de la documentation n'est pas parfaite. Il est possible que cet écart de valeur soit également dû au matériel et aux conditions d'environnement. Cependant, les $150\mu s + 47\mu s = 197\mu s$ de durée de démarrage donnée dans la documentation, et les $120\mu s + 40\mu s + 30\mu s = 190\mu s$ relevées dans nos observations donnent une durée de démarrage approximativement similaire, corroborant ainsi nos suppositions. En suivant l'ordonnancement de la séquence de démarrage de la figure 5.3, nous focalisons nos observations sur la fin de cette période de démarrage. Le *Zoom* 2.1 de la figure 5.7 représentant la période du signal analysée, est donné par la figure 5.8. Sur celle-ci, une divergence de comportement est observée en fonction de l'état de verrouillage du débogage.

On estime que l'instant mis en valeur fait parti d'une phase de fonctionnement du contrôleur d'accès au débogage. La présence d'une divergence de comportement en fonction de l'état de verrouillage durant cette période permet de définir une fenêtre temporelle durant laquelle une injection EM pourrait perturber le mécanisme de verrouillage.

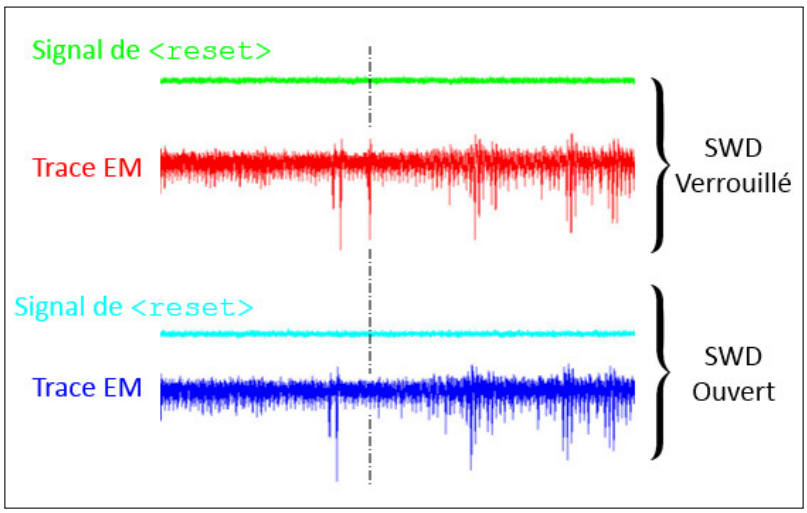


FIGURE 5.8 – Zoom 2.1 de la Fig.5.7. Au niveau du repère en pointillé, un pic présent sur la trace mesurée lorsque le débogage est verrouillé indique une divergence de comportement en fonction de l'état de verrouillage du débogage.

5.3.4 Perturbation du mécanisme de verrouillage

L'analyse des vulnérabilités a mis en valeur des opérations sensibles susceptibles d'être détournées par l'injection d'une perturbation EM. La période d'exécution de ces opérations, et donc l'instant t_{inj} auquel une perturbation doit être injectée, a été défini par l'analyse *side-channel*.

5.3.4.1 Procédure d'injection de faute

On considère le microcontrôleur dont la flash contient un micrologiciel et dont l'accès au débogage est verrouillé. Pour cela, avec la sonde de débogage, nous modifions la valeur du « statut de verrouillage du débogage » dans la mémoire flash. En inscrivant `0xBADA_BADA`, une valeur différente de `0xFFFF_FFFF` et facilement distinguable dans une page mémoire emplie de '`FF`', et en redémarrant le microcontrôleur, on verrouille l'accès au débogage sans effacer de donnée. La procédure de test consiste à envoyer un signal de `<reset>` pour redémarrer le microcontrôleur et de générer une impulsion EM. Puis avec la sonde de débogage, une requête qui demande l'IDCODE du AHB-AP est envoyée. Si la réponse du microcontrôleur est la valeur demandée, alors cela signifie que l'attaque a réussi et que le débogage est déverrouillé. En revanche, si la valeur retournée est celle du contrôleur d'accès, alors le débogage est toujours verrouillé. Cette séquence est répétée en balayant une fenêtre temporelle autour de l'instant de divergence de comportement t_{inj} mis en avant dans l'analyse précédente.

Pour augmenter la probabilité de succès de l'attaque, deux paramètres ont été pris en compte : la présence d'une éventuelle désynchronisation due à du *jitter* et l'incertitude de l'instant précis à perturber. Pour cela, deux solutions ont été apportées. La première concerne le déclenchement des perturbations EM. Afin d'en accroître la précision, elles

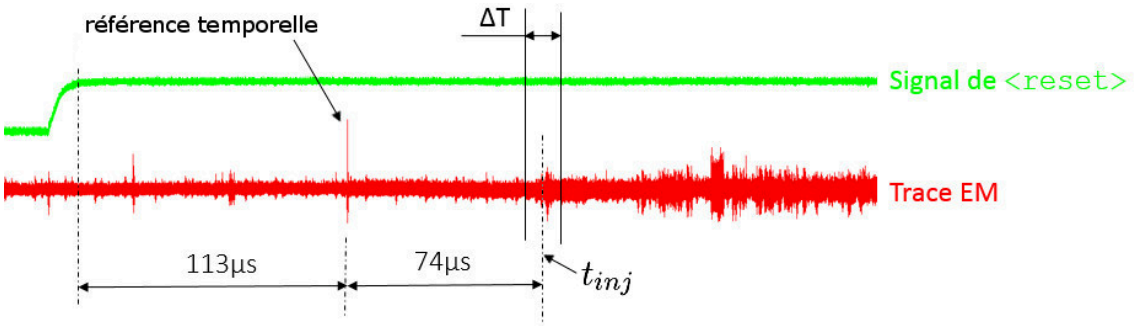


FIGURE 5.9 – Balayage temporel autour de l'instant mis en valeur par l'analyse *side-channel*. Les mesures EM ont été effectuées avec la sonde d'observation du banc d'injection EM.

sont générées en prenant comme référence un motif du signal EM synchronisé avec la zone ciblée. Ainsi un pic toujours distant de $74\mu s$ est pris comme référence temporelle. Cette première mesure demande l'utilisation d'une sonde insensible aux impulsions EM (voir (8) de la Section 2.5 du Ch.2). La seconde solution concerne la localisation précise de la zone temporelle à perturber. Pour ajuster la valeur de l'instant susceptible de produire le comportement espéré, les injections sont effectuées selon un balayage temporel sur une fenêtre ΔT. La figure 5.9 illustre ces deux solutions.

L'ensemble de la surface du microcontrôleur a été balayé avec le banc d'injection EM (voir Ch.2, Section 2.5). Une recherche spatio-temporelle est effectuée, augmentant certes le nombre d'essais mais aussi les chances de succès. Pour chaque point spatial, les injections EM sont effectuées dans la fenêtre $\Delta T = [\![t_{inj} - 2500ns, t_{inj} + 2500ns]\!]$ avec un pas de $100ns$ entre chaque essai. Les résultats présentés sont ceux obtenus avec l'injecteur EM Cyl.-∅800-N7 (voir (2), Section 2.5 du Ch.2). L'amplitude et la durée d'injection sont respectivement configurées à $A_{inj} = 280V$ et $\Delta T_{inj} = 10ns$. La figure 5.10 présente l'orientation de la zone de balayage considérée à la surface du contrôleur ainsi que la sonde et l'injecteur EM utilisés.

5.3.4.2 Résultats expérimentaux

La surface du microcontrôleur considérée a été balayée en un peu plus de 4 jours. Sur les 520251 injections EM, l'IDCODE du contrôleur AHB-AP a été retourné deux fois au-dessus du même point spatial. Les temps d'injection pour lesquels l'attaque a réussi sont respectivement $74.600\mu s$ et $75.200\mu s$. Pour casser la sécurité du système, une seule attaque réussie suffit. Cependant, afin d'analyser l'effet de notre perturbation, nous avons effectué une seconde campagne d'injection en fixant spatialement l'injecteur EM au dessus du point pour lequel les deux IDCODE ont été renvoyés. En balayant temporellement la fenêtre $\Delta T = [74.500\mu s; 75,500\mu s]$ avec un pas de $100ns$ et 10 essais à chaque instant, le taux de réussite a été porté à près de 10%.

Ceci nous a permis de reproduire l'attaque pour analyser plus précisément l'effet produit par la perturbation EM. Le script qui exécute la séquence de test a été paramétré pour être

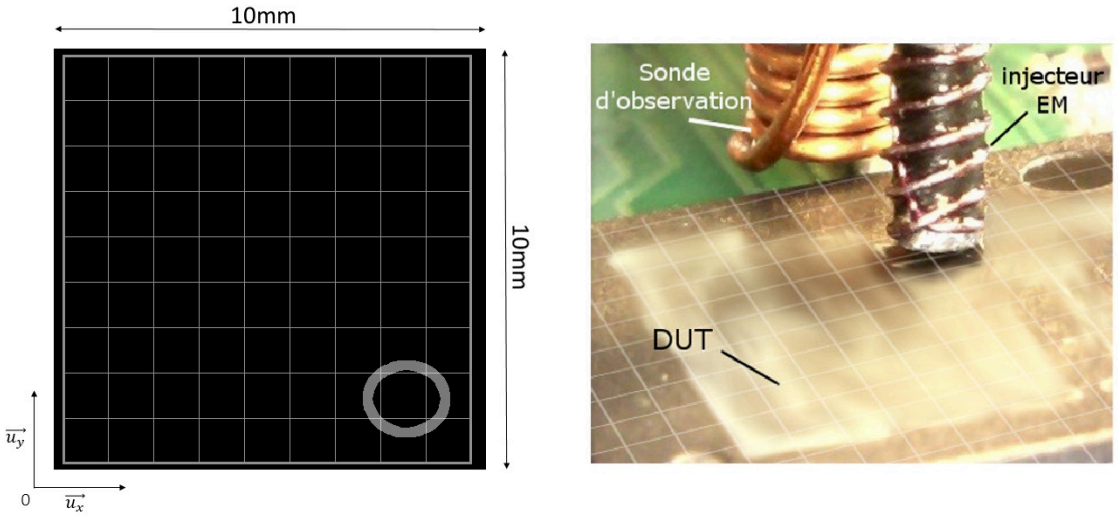


FIGURE 5.10 – Configuration des expérimentations d'injection EM. L'image de gauche donne l'orientation de la zone considérée pour la cartographie et celle de droite montre la sonde d'observation et l'injecteur utilisés.

interrompu dès qu'une attaque réussit et avant le redémarrage du microcontrôleur. Ainsi nous avons pu utiliser l'interface de débogage SWD pour aller lire l'état de la mémoire. Nous avons constaté que les données y étaient toujours présentes dans leur intégralité, et en particulier le « statut de verrouillage du débogage » qui a toujours la valeur `0xBADA_BADA`. Ceci confirme que nous avons perturbé la lecture du statut de verrouillage sans modifier de valeur. Etant donné que le « statut de verrouillage du débogage » est différent de sa valeur initiale, un redémarrage du contrôleur rétablirait le verrouillage. Cependant, l'accès au débogage que nous avons obtenu temporairement nous permettrait d'en déverrouiller l'accès de façon permanente, si nous le souhaitions.

5.3.5 Etude de faisabilité d'attaque des mécanismes de verrouillage : conclusion

L'analyse de la documentation constructeur, disponible librement sur internet, a permis de cerner deux processus critiques vis-à-vis du verrouillage du débogage. C'est en effectuant une analyse *side-channel* précise du fonctionnement de ce verrouillage qu'il a été possible de localiser l'instant d'exécution des processus susceptibles d'être perturbés. Avec un balayage spatio-temporel conséquent mais raisonnable, nous avons testé la résistance de ces processus face aux perturbations EM. Le déverrouillage du débogage sans effacer l'intégralité des données a été observé durant cette première cartographie. Puis, un ajustement des paramètres d'injection a permis d'augmenter le taux de réussite de l'attaque à près d'un essai sur dix, démontrant ainsi la présence d'une faille sécuritaire importante dans ce mécanisme. Sans documentation détaillée de l'architecture du contrôleur d'accès au débogage et sur son fonctionnement, il ne nous a pas été possible d'expliquer l'origine de la vulnérabilité. Cependant, nous pouvons conclure que, pour ce microcontrôleur, le dispositif

de verrouillage du débogage n'est pas résistant aux attaques matérielles.

Dans la suite de ce chapitre, nous allons présenter une nouvelle utilisation détournée du débogage matériel pour effectuer des attaques. Deux exemples illustrent nos propos, mais les opportunités proposées par le débogage matériel permettent d'imaginer une multitude d'attaques complexes.

5.4 Utilisation du débogage matériel comme vecteur d'injection de faute

Cette partie présente les nouvelles possibilités d'attaques offertes par les outils de débogage matériel lorsqu'ils disposent d'un accès libre aux ressources des SoC. L'utilisation impropre du débogage matériel n'est pas nouvelle mais, jusqu'à présent, la littérature l'a présenté seulement comme un moyen d'extraire des donnés [119, 1]. Cependant, de tels outils ont les capacités d'accéder aux ressources et aux fonctionnalités internes des systèmes. Ils peuvent interrompre des processus et en modifier les données. Ceci met en valeur le fait que nous avons à notre disposition un vecteur de perturbation capable d'injecter de manière très précise une erreur dans un programme en cours d'exécution au sein d'un SoC. Aucune grandeur physique habituellement utilisée pour perturber un système ne permet d'avoir une précision aussi élevée ainsi qu'un tel contrôle. Ainsi, dans la Section 5.4.2, nous proposons d'utiliser le débogage matériel non pas pour extraire des données en mémoire mais plutôt comme moyen extrêmement précis d'injection de faute. Pour illustrer cette nouvelle utilisation, nous présentons deux applications du JTAG dans des scénarios d'attaques. Le premier est une attaque par injection de faute visant à dégrader le niveau de sécurité d'une fonction système (Section 5.4.3). Le second scénario présente l'utilisation du JTAG dans une attaque combinée pour une élévation de privilège (Section 5.4.4). Ces deux applications exploitent les fonctionnalités du JTAG, mais l'injection de fautes est applicable au mécanisme SWD.

5.4.1 Contexte expérimental

Les deux applications du JTAG présentées dans cette section ont volontairement été effectuées sur des systèmes complexes afin de démontrer les possibilités d'attaques offertes par les outils de débogage. Deux SoC d'architectures différentes ont été sélectionnés en tant que véhicules de test. Ils sont décrits dans les sections expliquant les expérimentations.

5.4.1.1 Le système d'exploitation

« Android » est le système d'exploitation utilisé par près de 80% des téléphones mobiles du marché. Cet OS est basé sur un noyau Linux et distribué en *open source*. Ceci permet d'une part, d'avoir accès au code source, et d'autre part, de garantir une large communauté active de développeurs. Ces derniers analysent et proposent constamment des améliorations sur les forums et les sites internet dédiés. En particulier, les vulnérabilités détectées sont publiées, ce qui permet de les corriger, mais aussi de s'en inspirer. Les deux SoC de nos expérimentations fonctionnent avec la version Android Marshmallow 6.0.1[20].

5.4.1.2 Android Debug Bridge (ADB)

Android Debug Bridge est un outil polyvalent en ligne de commande qui permet de communiquer avec une instance d'Android, que celle-ci s'exécute sur une machine virtuelle ou réelle. Les fonctionnalités proposées par ADB permettent une multitude d'actions envers le système ciblé, comme installer ou déboguer des applications. Cet outil permet également

20. https://www.android.com/versions/marshmallow-6-0/

un accès à la couche Linux d'Android au travers d'un `shell` Unix et d'utiliser un ensemble restreint de commandes. Comme représenté sur la figure 5.11, ADB est un programme fonctionnant en mode client-serveur, constitué de trois éléments :

- **Un client**, qui envoie les commandes. Le client s'exécute sur la machine de développement. Il peut être invoqué depuis un terminal, en exécutant une commande ADB ou depuis le SDK d'Android.
- **Un démon (adbd)**, qui s'exécute en arrière-plan sur la machine Android. Chaque instance d'Android a un démon en arrière-plan.
- **Un serveur**, qui gère les communications entre le client et le démon. Le serveur fonctionne lui aussi en arrière-plan mais sur la machine de développement.

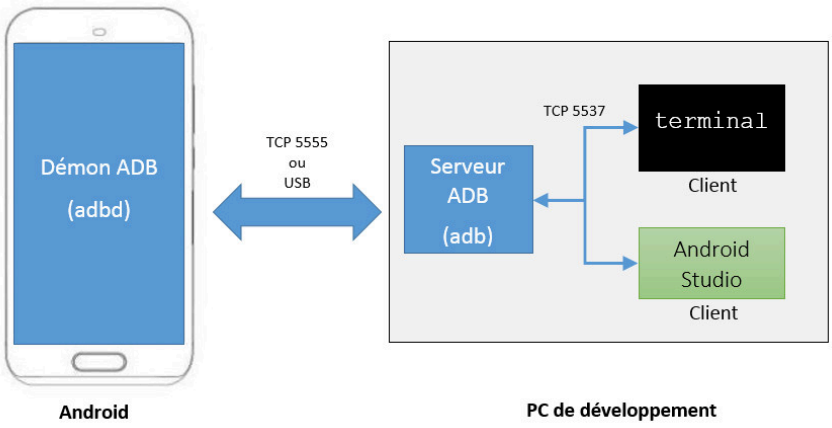


FIGURE 5.11 – Schéma client-serveur d'Android Debug Bridge

Lorsqu'une commande ADB est lancée dans un terminal, le client vérifie si un processus serveur est démarré. Si ce n'est pas le cas, le client démarre le processus serveur qui va écouter les commandes envoyées par tous les clients sur le port TCP 5037. Puis le serveur établit une connexion avec tous les appareils Android qu'il détecte.

5.4.1.3 Outil de débogage matériel

L'outil de débogage utilisé dans les manipulations présentées dans cette section, est la sonde PowerDebug PRO de Lauterbach[21]. Le logiciel Trace32 de cette sonde propose une interface graphique très complète, conçue pour donner à l'utilisateur une vue globale du système et lui permettre d'effectuer un grand nombre d'opérations sur celui-ci. Il est possible, en particulier, d'arrêter le CPU, d'accéder à la mémoire et d'y modifier des valeurs. Une série d'API permet d'intégrer le fonctionnement de la sonde dans un programme C. De cette manière, les routines de débogage peuvent être automatisées, comme l'ont demandé les expérimentations qui suivent.

21. https://www.lauterbach.com/

5.4.2 Le débogage matériel comme vecteur d'injection de faute

Dans un programme ou un système d'exploitation, une fonction f impliquée dans une opération de sécurité est ciblée. Soit $R = \{r_0, \ldots, r_n\}$, un ensemble de règles déterminant le comportement de la fonction notée f_R. Dans un premier temps, comme pour chaque attaque par injection de faute, une analyse de vulnérabilité est effectuée. Cette analyse cherche à identifier les règles $r_i{}^* \in R$ impliquées dans la sécurité basée sur un test conditionnel entre une valeur v et une valeur de référence v_{ref}. Puis, avec un outil de débogage matériel (JTAG ou SWD), l'exécution de la fonction f_R est stoppée avant que ne soit appliquée la règle $r_i{}^*$. Les capacités de débogage de l'outil lui permettent d'accéder et de modifier dans la mémoire, soit à la valeur testée v, soit à la valeur de référence v_{ref}. Tout dépend de l'implémentation du code ciblé et du scénario de l'attaque :

1. **Modification de la valeur testée v.** Lorsque la valeur ciblée est la valeur testée v, la règle $r_i{}^*$ est appliquée mais le test conditionnel devient arbitraire puisqu'il est manipulable par l'attaquant. En effet, en modifiant v en v_{ref} ou $\neq v_{ref}$, le test conditionnel pourra au choix être validé ou pas, ce qui permettra d'exploiter les processus protégés par le test. Par conséquent, f_R est détournée ce qui compromet la sécurité du système.

2. **Modification de la valeur de référence v_{ref}.** Lorsque c'est la valeur de référence v_{ref} qui est modifiée en une valeur différente v'_{ref}, la règle définie sur cette valeur change de $r_i{}^*$ en $r_i{}^{*\prime}$ impliquant que $R \longrightarrow R'$. Le résultat est similaire à la modification précédente et la fonction $f_{R'}$ n'applique plus les règles de sécurité initiales.

Dans les deux cas, le test conditionnel ne remplit plus sa fonction de contrôle et est à la merci de l'opérateur de la sonde de débogage. Le principe donné ici de manière générique, peut être illustré par le module de verrouillage de l'accès au débogage étudié dans la section 5.3 de ce chapitre. Avec cet exemple, la fonction f_R est effectuée par le contrôleur d'accès au débogage , la valeur de référence v_{ref} est `0xFFFF_FFFF` et la valeur testée v est la valeur dans la mémoire flash. Avec ces considérations, la règle $r_i{}^*$ peut être formulée par : « Si v est égale à v_{ref}, alors le débogage est ouvert ». Les moyens de contourner cette règle sont de modifier v ou v_{ref}. Dans le cas de ce contrôleur, la modification de v (modification n° 1) avec la sonde SWD est plus simple.

5.4.3 Application 1 : Injection de faute avec le JTAG pour dégrader le niveau de sécurité d'une fonction système

Dans cette partie, pour illustrer l'utilisation d'un outil de débogage comme moyen d'injection de fautes, nous présentons une attaque visant à dégrader le niveau de sécurité d'une fonction système. Plus précisément, nous attaquons la fonction qui gère l'affichage des adresses des fonctions système de l'OS Android. Par sécurité [22], lorsqu'un utilisateur sans privilège essaye de lire ces adresses, la fonction en charge de contrôler cette information retourne une série de `'00...0'`.

22. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=455cd5ab305c90ffc422dd2e0fb634730942b257

5.4.3.1 Principe de l'attaque

Soit f_R, la fonction affichant les adresses des fonctions système. Soit $r_i{}^* \in R$, la règle de sécurité qui détermine le niveau de privilège requis pour afficher les adresses. Par définition $\forall v \in \mathbb{N},\ r_i{}^*(v)$:

- $r_i{}^*(v = 0)$: « Afficher les adresses à tous les utilisateurs ».
- $r_i{}^*(v = 1)$: « Afficher les adresses uniquement pour les utilisateurs privilégiés ».
- $r_i{}^*(v \geq 2)$: « Ne jamais afficher les adresses ».

Ici, un utilisateur est une entité susceptible de demander l'information des valeurs d'adresses (utilisateur humain, un programme, une fonction, etc.). Dans un système Android fonctionnant dans ses conditions nominales, par défaut $v = 1$.

Le principe de notre attaque est d'utiliser la sonde de débogage matérielle pour localiser la valeur v dans la mémoire et la forcer à '0'. Il s'agit du scénario d'attaque qui modifie la valeur testée v (voir 1. Section 5.4.2).

5.4.3.2 Précisions techniques

Afin d'expliquer clairement l'attaque et de comprendre les difficultés auxquelles nous avons dû faire face, il est nécessaire de donner des précisions techniques concernant Android.

Dans les systèmes d'exploitation Android, et plus généralement dans les systèmes basés sur Linux, les adresses des symboles du noyau sont affichées par le fichier `/proc/kallsyms`. Cependant, afficher ces pointeurs d'adresses fournit un moyen aux attaquants de détecter facilement les vulnérabilités en écriture du noyau. En effet, ces pointeurs révèlent les emplacements de structures inscriptibles contenant des pointeurs de fonctions facilement appelables. Pour des raisons de sécurité, la fonction `s_show` ($\equiv f_R$) du fichier `kallsyms` utilise des *format specifiers* `%pK` ($\equiv r_i{}^*$) conçus pour dissimuler les pointeurs du noyau, notamment via les interfaces du fichier `/proc`. Le comportement de `%pk` dépend d'une valeur entière contenue dans le *sysctl* `kptr_restrict` ($\equiv v$). Par définition dans le code source du noyau Linux [23] :

- `kptr_restrict` $= 0$: « Aucune restriction d'affichage ».
- `kptr_restrict` $= 1$: « Les pointeurs du noyau utilisant `%pk` sont remplacés par des '0', sauf si l'utilisateur possède les privilèges qui lui donnent la permission de les afficher ».
- `kptr_restrict` ≥ 2 : « Quel que soit le niveau de privilège de l'utilisateur, les `%pk` sont remplacés par des '0' ».

La plupart des systèmes Android du commerce sont en mode utilisateur et la valeur de l'entier `kptr_restrict` vaut '1'. Cette valeur ne peut être modifiée que par un utilisateur privilégié.

Avec ces précisions techniques, le principe de l'attaque est de forcer la valeur de `kptr_restrict` à '0' avec l'outil de débogage matériel.

23. https://www.kernel.org/

5.4.3.3 Expérimentations

5.4.3.3.1 Véhicule de test.

Les expérimentations ont été effectuées sur une carte de développement embarquant un SoC très répandu. Il s'agit d'un système d'architecture ARMv8 64 bits composé de quatre cœurs ARM Cortex-A72 cadencés jusqu'à une fréquence de 2,3GHz, et quatre cœurs Cortex-A53 pouvant fonctionner jusqu'à 1,8GHz. Le système d'exploitation est Android 6.0.1 - Marshmallow. Pour nos expérimentations, la carte de développement a volontairement été laissée en mode utilisateur. Pour communiquer avec Android, nous avons utilisé ADB sans privilège, en simple utilisateur.

5.4.3.4 Procédure de l'attaque

Le succès de cette attaque dépend de la capacité à modifier la valeur contenue dans `kptr_restrict`. La principale difficulté est de trouver l'adresse de cette valeur dans la mémoire. En l'occurrence, la complexité de l'attaque est due à la taille de la mémoire principale et la gestion des adresses 64 bits par l'OS du SoC.

Le système d'exploitation chargé dans la RAM peut être représenté par des valeurs hexadécimales issues de la compilation d'un code. Ces valeurs dépendent principalement du compilateur utilisé et des options de compilation. Afin de localiser la valeur contenue dans `kptr_restrict`, par analogie avec la méthode de localisation temporelle proposée Ch.2, Section 2.3.1.2, des marqueurs ont été utilisés pour encadrer la zone d'adresse contenant la valeur ciblée. Les marqueurs utilisés sont des chaînes de caractères ASCII facilement identifiables, déjà présentes dans le code. Par conséquent, la procédure d'injection de faute avec le JTAG peut se décomposer en deux étapes principales :

1. Avec l'outil de débogage JTAG, localiser l'adresse d'une chaîne de caractère ASCII particulière, proche de l'adresse de `kptr_restrict`.

2. A partir de cette adresse, aller à l'adresse de `kptr_restrict` en changant successivement les valeurs rencontrées et en vérifiant systématiquement si les pointeurs d'adresses sont différents de '`0000000000000000`' dans le fichier `/proc/kallsyms`.

Les sections suivantes décrivent la manière dont ces deux points ont été abordés et comment les difficultés rencontrées ont été résolues.

5.4.3.4.1 Localiser l'emplacement à perturber

La valeur de `kptr_restrict` est un entier représenté par un mot de 32 bits dans la section `.data` de la mémoire du système. Puisque les adresses des symboles du noyau sont remplacées par des '0' dans le véhicule de test, alors sa valeur est soit `0x1`, soit `0x2`. Dans un processeur moderne d'architecture 64 bits, localiser directement quel `0x1` ou `0x2` correspond à la valeur de `kptr_restrict` parmi les $2^{64} - 1$ adresses mémoires serait une tâche longue et fastidieuse. Pour donner un ordre de grandeur, sur 2,1Go de données copiées depuis la mémoire avec le JTAG, on dénombre 1733598 `0x1` et 1349512 `0x2`. Vérifier l'effet du changement de chacune de ces valeurs en `0x0` sur la lecture des adresses serait vraiment consommateur de temps.

```
user@android:/home/user/ $ cat /proc/kallsyms
                           ⋮
0000000000000000 D kptr_restrict
                           ⋮
0000000000000000 d pfifo_fast_ops
0000000000000000 D noqueue_qdisc_ops
0000000000000000 D noop_qdisc_ops
                           ⋮
```

FIGURE 5.12 – Exécuter la commande `"cat"` dans un `shell` Unix au travers l'interface d'ADB, permet d'afficher l'ordonnancement des adresses masquées dans le champs `.data` de la mémoire.

Cependant, le champ `.data` de la mémoire du noyau Linux contient également des chaînes de caractères ASCII que nous considérons comme des marqueurs. Comme il est détaillé dans plusieurs des fichiers du noyau Linux [24], une partie de ces chaînes sont des champs `.id` de structures. En particulier, les champs contenus dans les fichiers `sch_generic.c` et `sch_fifo.c` du répertoire `/net/sched`, qui sont les chaînes de caractères stockées juste après la valeur de `kptr_restrict` dans la mémoire. Afficher les adresses des symboles remplacées avec des '0' nous permet malgré tout de connaître leur agencement dans la mémoire, comme représenté Fig. 5.12.

Mais, ici aussi, rechercher des chaînes de caractères dans toute la mémoire d'un système d'addresse à 64 bits, représente une quantité de travail non négligeable. C'est pourquoi une astuce est utilisée pour localiser approximativement une adresse à laquelle commencer les recherches. Sur tous les systèmes Android et pour tous les utilisateurs même non privilégiés, les versions du noyau et du compilateur utilisés pour construire l'OS peuvent être connues en entrant la ligne de commande `adb shell cat /proc/version` au travers de l'interface ADB. Dans cette expérimentation, la version du noyau Linux est la `4.1.15` et le compilateur est `gcc version 4.9.x-Google`. En téléchargeant la même version de noyau [25] et le même compilateur, il est possible de construire le fichier `System.map` qui fournit une cartographie

24. https://github.com/torvalds/linux
25. https://www.kernel.org/

de la mémoire décalée d'un éventuel offset dû à des options de compilation différentes. Ce fichier fournit les mêmes informations que le fichier `/proc/kallsyms` sans que les adresses ne soient masquées par des '0'. L'adresse du symbole `_text` permet de définir les adresses des blocs mémoires réservés aux sections `.text` et `.data` du noyau. L'adresse $@_{SD}$ = `0xFFFF_FFC0_0000_0000` est le début de la section `.data` dans laquelle se trouvent les valeurs recherchées. Ainsi nous procédons de la sorte :

1. Avec le JTAG, à partir de $@_{SD}$, extraire P, une portion de données mémoire de taille raisonnable (quelques ko).

2. Analyser P afin de repérer les chaînes de caractères ASCII servant de marqueurs. Si toutes les chaînes sont identifiées, passer à l'étape suivante. Sinon, retourner à l'étape 1. en extrayant une nouvelle portion de données de la même taille que P, à partir de l'adresse $@_{SD}$ + (taille de P). Procéder ainsi jusqu'à identifier toutes les chaînes de caractères recherchées, dans l'ordre donné par `/proc/kallsyms`.

Si on considère les chaînes de caractères des champs `.id` des structures contenues dans les fichiers `sch_generic.c` et `sch_fifo.c`, les valeurs recherchées sont celles figurant dans le tableau 5.2. Il faut tenir compte du fait que le processeur que nous ciblons code ses données hexadécimales en mode *little-endian* (octet de poids le plus fort à l'adresse la plus faible) sur des mots de 32-bits.

Chaîne de caractère	Valeur ASCII en hexadécimale
`"pfifo_fast"`	`66696670 61665F6F 00007473`
`"noqueue"`	`75716F6E 00657565`
`"noop"`	`706F6F6E`
`"bfifo"`	`66696662 0000006F`
`"pfifo_head_drop"`	`66696670 65685F6F 645F6461 00706F72`
`"pfifo"`	`66696670 0000006F`

Tableau 5.2 – Valeurs ASCII en notation hexadécimales *little-endian* des chaînes de caractères recherchées dans la mémoire, dans cet ordre.

Ainsi, en utilisant le même principe que celui des algorithmes de reconnaissance des formes, des portions de données ont été extraites de la mémoire et analysées jusqu'à ce que les valeurs recherchées soient localisées dans le même ordre que celui du tableau 5.2. Cela représente un ensemble de 27 portions de 78ko de données chacune, extraites et analysées en approximativement 52s. Le tableau 5.3 liste les adresses de chaque chaîne de caractères identifiée dans la mémoire et la figure 5.13 illustre leur position relative. Seule la chaîne `"noop"` apparaît plusieurs fois avant que toutes les chaînes ne soient trouvées dans le bon ordre. L'adresse du *sysctl* `kptr_restrict` étant inférieure à celle de ces chaînes de caractères, c'est l'adresse de la chaîne `"pfifo_fast"` : $@_{\texttt{pfifo\_fast}}$ = `0xFFFF_FFC0_00D6_CD88` qui sert de point de départ pour la prochaine étape.

Chaîne de caractère	Localisé à l'adresse :
"pfifo_fast"	0xFFFF_FFC0_00D6_CD88
"noqueue"	0xFFFF_FFC0_00D6_CE10
"noop"	0xFFFF_FFC0_009E_76B0 0xFFFF_FFC0_00A1_5ECC 0xFFFF_FFC0_00C0_E2F0 0xFFFF_FFC0_00D6_CE98
"bfifo"	0xFFFF_FFC0_00D6_D030
"pfifo_head_drop"	0xFFFF_FFC0_00D6_D0B8
"pfifo"	0xFFFF_FFC0_00D6_D140

Tableau 5.3 – Liste des adresses des valeurs ASCII en notation hexadécimale des chaînes de caractères trouvées dans la mémoire à partir de de l'adresse $@_{SD}$ = 0xFFFF_FFC0_0000_0000

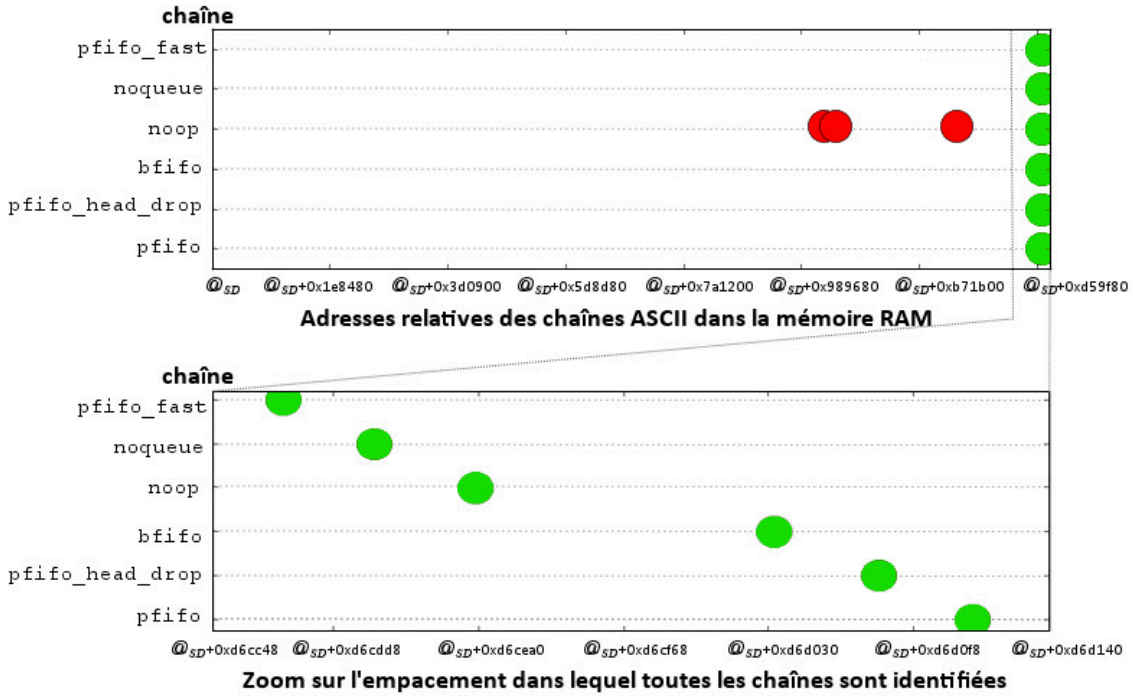


FIGURE 5.13 – Cartographie mémoire. Position relative des adresses des chaînes de caractères recherchées dans la mémoire RAM

5.4.3.4.2 Injecter la faute

Cette dernière étape est un balayage d'injections de fautes jusqu'à l'obtention de l'effet escompté. La sonde JTAG va balayer et modifier les valeurs directement dans la mémoire RAM à partir de l'adresse $@_{\texttt{pfifo\_fast}}$, jusqu'à l'adresse de `kptr_restrict`. Cela représente un maximum d'environ 2Mo à parcourir depuis $@_{\texttt{pfifo\_fast}}$ jusqu'au début de la section `.data` à l'adresse $@_{SD}$. Cette recherche de valeur est décrite par la procédure suivante :

(init) `addr` = `0xFFFF_FFC0_00D6_CD88` (l'adresse de la chaîne `"pfifo_fast"`)

(1) `addr` = `addr - 0x4`

(2) Avec la sonde JTAG, lire la valeur à l'adresse `addr`. Si cette valeur est égale à `0x0000_0001` ou `0x0000_0002` alors la changer en `0x0000_0000`. Puis avec l'interface ADB, envoyer la ligne de commande pour vérifier si les adresses des symboles du noyau sont disponibles : `adb shell cat /proc/kallsyms`. Si les valeurs retournées sont masquées par des '0' alors il faut rétablir la valeur initialement stockée à l'adresse `addr` et lire la valeur contenue à l'adresse suivante *i.e.* retourner à l'étape **(1)**. Autrement, l'attaque a réussi et les symboles du noyau sont accessibles à un simple utilisateur.

5.4.3.5 Résultats expérimentaux

Durant l'exécution de la routine précédente, seulement six `0x0000_0001` et un unique `0x0000_0002` ont été dénombrés. La valeur du `kptr_restrict` est le `0x0000_0002` trouvé à l'adresse `0xFFFF_FFC0_00D6_B320`.

5.4.4 Application 2 : le JTAG dans une attaque combinée pour élever les privilèges

La partie précédente a montré la capacité d'injecter précisément une erreur dans un code s'exécutant dans un SoC. Dans cette partie, nous utilisons cette capacité pour présenter l'exploitation du JTAG dans une attaque combinée qui a pour objectif une élévation de privilèges. Ceci témoigne de la possibilité d'élaborer des attaques complexes avec un tel outil.

5.4.4.1 Attaque par élévation de privilèges

Dans [2], Davi *et al.* définissent une attaque par élévation de privilèges comme étant la cause permettant à une application sans privilège d'accéder, sans aucune restriction, à des ressources réservées aux applications privilégiées. Plus généralement, l'élévation de privilèges est une action qui exploite un bug ou un défaut de conception afin de contourner un processus de contrôle d'accès. Le résultat d'une telle attaque est qu'un utilisateur obtient des privilèges non prévus par les concepteurs, lui permettant d'effectuer des actions qui initialement lui étaient interdites. Avoir des privilèges distribués arbitrairement à des utilisateurs, leur donnant ainsi, accès aux données garantes de la fiabilité du système, représente un risque majeur. Il s'agit de l'un des types d'attaques les plus redoutés dans le domaine de la sécurité numérique.

5.4.4.2 Origine de l'attaque

L'utilisation du débogage matériel pour perpétrer une attaque d'élévation de privilèges a été inspirée par une attaque exclusivement logicielle qui exploite un défaut de paramétrage de ressources matérielles. Pour comprendre notre attaque, il est nécessaire de présenter les principes de cette attaque logicielle.

5.4.4.2.1 « Samsung Exynos Kernel Exploit »

En décembre 2012, le pirate éthique de la communauté Android connu sous le pseudonyme Alephzain, publie une méthode pour obtenir les privilèges *root* sur la plupart des téléphones Samsung du moment [12]. Sa méthode utilise un exécutable et s'applique sur tous les appareils Samsung intégrant un processeur Exynos 4210 ou 4412. La vulnérabilité exploitée est une erreur de paramétrage des privilèges nécessaires pour accéder au fichier `/dev/exynos-mem`. Il s'agit du ficher système qui permet d'accéder à toute la mémoire physique du système. En l'occurrence, sa configuration était laissée en mode lisible et inscriptible pour tous les utilisateurs, privilégiés ou non. Il s'agit là d'une grave négligence de la part des concepteurs. Afin d'alerter Samsung et la communauté, Alephzain a proposé un exemple d'attaque par élévation de privilège pour illustrer les possibilités offertes par une telle vulnérabilité.

5.4.4.2.2 Principe de l'attaque logicielle

La fonction système `sys_setresuid`[26] permet de modifier les identifiants des utilisateurs (UID) de la couche Linux d'Android. Ces UID sont principalement utilisés pour contrôler les droits d'accès aux ressources et aux domaines du système d'exploitation. Le super-utilisateur qui possède tous les privilèges, a l'UID 0. Seul ce dernier peut demander à la fonction `sys_setresuid` d'octroyer l'UID 0 à d'autres processus. Par conséquent, un utilisateur non privilégié ne pourra pas allouer un UID procurant plus de privilèges qu'il n'en a à un autre processus.

Avec la possibilité de lire ou d'écrire à n'importe quelle adresse dans la mémoire physique du système grâce à la vulnérabilité du fichier `/dev/exyns-mem`, Alephzain montre qu'il est possible de détourner la fonction `sys_setresuid`. Il a implémenté une application sans privilèges qui modifie le code de cette fonction dans la mémoire pour que cette dernière attribue l'UID 0 à tout les utilisateurs qui le demandent. C'est une opération de comparaison qui est modifiée : `"cmp r0,#0x0"` en code assembleur ARMv7. Nous supposons que cette opération tient compte des privilèges de l'utilisateur appelant la fonction, pour valider l'attribution d'une valeur d'UID. La comparaison est modifiée en `"cmp r0,#0x1"` ce qui change complètement son comportement. Etant donné que l'exécutable manipule directement les valeurs du code machine dans la mémoire, la valeur est modifiée de `"0xE3500000"` en `"0xE3500001"`. Avec cette modification, si un simple utilisateur ou un processus demande à la fonction `sys_setresuid` de se faire octroyer l'UID à 0, il l'obtiendra sans difficulté. Dans son exemple, après avoir modifié la comparaison de la fonction dans la mémoire, le code d'Alephzain demande à ouvrir un terminal privilégié :

26. http://man7.org/linux/man-pages/man2/setresuid.2.html

un `shell root`. Cette manipulation permet à une application de s'allouer des privilèges qu'elle n'aurait pas dû avoir. C'est une élévation de privilèges.

Depuis la publication de cette attaque, Samsung a comblé cette faille par une mise à jour de ses logiciels, et désormais la vulnérabilité `/dev/exynos-mem` n'existe plus. Cette de vulnérabilité n'a jamais été observée sur d'autres appareils utilisant Android. Cependant, si un attaquant a un accès à la mémoire physique d'un processeur fonctionnant sous Android, le schéma d'attaque est toujours valide. C'est ce que nous présentons dans la partie suivante.

5.4.4.3 Principe de notre attaque combinée

En considérant le principe de l'attaque Exynos et les capacités offertes par les outils de débogage matériel, nous montrons qu'il existe un autre chemin pour accéder directement aux données contenues dans la mémoires d'un système. Lorsque le débogage matériel est ouvert sur un appareil Android, rien n'empêche un attaquant d'aller lire et modifier sa mémoire pour procéder à une élévation de privilèges. Ainsi, sur le principe de l'attaque Exynos, nous proposons d'effectuer une élévation de privilèges par attaque combinée en utilisant le JTAG pour modifier la valeur dans la mémoire.

Avec les notations utilisées dans la Section 5.4.2, `sys_setresuid` est la fonction f_R qui attribue les valeurs des UID ($\equiv v$). La règle de sécurité $r_i{}^* \in R$, qui permet de contrôler l'allocation des privilèges, peut se résumer par $\forall v \in \mathbb{N}$, $v_{ref} = 0$, $r_i{}^*(v)$:

- $r_i{}^*(v = v_{ref})$: « Attribuer l'UID demandé, quelle que soit la valeur ».
- $r_i{}^*(v \neq v_{ref})$: « Attribuer l'UID demandé, excepté certaines valeurs réservées, en particulier l'UID 0 ».

La demande de modification d'UID peut être effectuée par un utilisateur ou un programme.

Le principe de notre attaque combinée est d'utiliser la sonde de débogage matérielle pour localiser, dans la mémoire, l'opération de comparaison avec la valeur v_{ref}, puis de changer v_{ref} en v'_{ref} (*i.e.* injecter une faute). Ainsi, nous transformons le comportement de la fonction en $f_{R'}$. Il s'agit du scénario d'attaque qui modifie la valeur de référence v_{ref} (voir 2. Section 5.4.2). Une fois la perturbation injectée, un exécutable demande les droits `root` au système.

5.4.4.4 Précisions techniques

Dans un système Linux, pour obtenir un `shell root`, au travers d'un terminal, un utilisateur va exécuter la commande `"sudo su"` en fournissant son mot de passe. Si cet utilisateur fait partie de la liste des utilisateurs autorisés à se voir octroyer les privilèges `root`, *i.e.* s'il fait partie des *sudoers*[27], alors un nouveau terminal avec le contexte `root` sera ouvert. Si cet utilisateur ne fait pas partie de la liste `/etc/sudoers`, alors un "`Operation not permitted`" est retourné.

Dans le contexte d'Android, un utilisateur communique avec la couche Linux du système au travers de l'interface fourni par ADB. Cette couche ne contient que très peu de binaires et le plus souvent `"su"` et `"sudo"` n'en font pas partie. De plus, le terminal proposé par

27. https://linux.die.net/man/5/sudoers

ADB n'est pas dans la liste des *sudoers*, donc, même si `"su"` et `"sudo"` étaient disponibles et exécutables dans la couche linux, ils n'auraient pas les permissions pour octroyer les privilèges `root` au terminal de l'interface ADB.

Pour contourner cela, comme Alephzain, nous utilisons un fichier exécutable qui est lancé depuis l'interface ADB. La figure 5.14 donne le code source de ce fichier. Celui-ci appelle la fonction système `sys_setresuid` au travers de la macro `setresuid` en demandant les droits `root` (ligne 15). Puis, avec la fonction système `execve`, un `shell` est lancé (ligne 21). Lorsque la fonction `sys_setresuid` a été modifiée et que les privilèges ont été octroyés à l'exécutable, le fait que ce dernier lance un `shell`, implique par hérédité de processus, que le `shell` est pourvu des mêmes privilèges, autrement dit des privilèges `root`.

```
1  /*
2   * run it for root shell when sys_setresuid is patched.
3   */
4
5  #include <unistd.h>
6
7  int main(int argc, char **argv, char **env) {
8
9      /* command for shell */
10     char *cmd[2];
11     cmd[0] = "/system/bin/sh";
12     cmd[1] = NULL;
13
14     /* ask for root permissions */
15     setresuid(0, 0, 0);
16
17     /* to be sure memory is updated */
18     usleep(100000);
19
20     /* execute a root shell */
21     execve (cmd[0], cmd, env);
22
23     return 0;
24 }
```

FIGURE 5.14 – Exécutable développé pour demander à la fonction `sys_setresuid` les privilèges `root` (UID 0) et ensuite ouvrir un `shell`

5.4.4.5 Expérimentations

5.4.4.5.1 Véhicule de test Les expérimentations ont été effectuées sur une carte de développement embarquant un SoC 32-bit d'architecure ARMv7. Ce système est composé de deux processeurs ARM Cortex-A9 cadencés jusqu'à une fréquence de 1,2GHz. La même version d'Android 6.0.1 - Marshmallow que l'expérimentation précédente s'exécute sur ces cœurs. Pour la pertinence de nos expérimentations, l'accès JTAG de la carte de dévelop-

pement est ouvert. Pour communiquer avec Android, nous utilisons ADB sans privilèges, en mode utilisateur simple.

5.4.4.6 Procédure de l'attaque

De la même manière que dans l'expérimentation précédente, le succès de cette attaque combinée ne dépend que de la capacité à modifier l'opération de comparaison de la fonction `sys_setresuid` dans la mémoire. La procédure de l'attaque combinée utilisant le JTAG comme vecteur d'injection de faute, se décompose en trois étapes :

1. Avec l'outil JTAG, localiser l'adresse de la fonction noyau `sys_setresuid`.
2. Modifier le code de la fonction pour détourner son comportement *i.e.* injecter la faute.
3. Depuis l'espace utilisateur d'ADB, demander les droits `root`.

Les sections suivantes décrivent comment ces trois points ont été mis en œuvres.

5.4.4.6.1 Localiser l'emplacement à perturber

Localiser le code de cette fonction dans la mémoire est l'une des tâches complexes de cette attaque. Avec les capacités de débogage du JTAG, trois possibilités s'offrent à nous :

- En utilisant le principe des algorithmes de reconnaissance des formes, nous pouvons chercher le code de la fonction `sys_setresuid` directement dans la partie mémoire réservée au code des fonctions du noyau.
- L'attaque de la Section 5.4.3 qui force le fichier `/proc/kallsyms` à révéler les pointeurs d'adresses des fonctions du noyau, conviendrait pour remplir cette tâche.
- Alephzain a dû faire face au même problème. Dans l'exécutable ayant servi pour son attaque, il utilise la possibilité d'accéder à la mémoire pour dégrader la sécurité de la fonction qui affiche les adresses. Ceci a pour conséquence de forcer `/proc/kallsyms` à révéler les pointeurs d'adresses.

La première méthode consiste à rechercher dans la mémoire, une suite de valeurs hexadécimales, qui correspond au code machine de la fonction `sys_setresuid`. Cependant l'ordonnancement et certaines valeurs hexadécimales dépendent du compilateur et des options qui ont été choisies lors de la génération du code machine. Par conséquent, la suite de valeurs recherchées dans la mémoire peut différer de la suite qui aura été prise comme référence. L'identification par reconnaissance de forme est compromise par cette caractéristique.

La seconde méthode est tout à fait applicable. Son efficacité a été démontrée dans la Section 5.4.3 de ce chapitre. Cependant, nous nous intéressons à l'utilisation des outils de débogage pour perturber un système opérationnel complexe. Afin d'explorer plusieurs possibilités d'utilisation, nous avons décidé d'adapter la méthode d'Alephzain à notre contexte d'attaque et de l'expérimenter.

Adaptation de la méthode logicielle à notre contexte.

Comme nous l'avons expliqué dans la Section 5.4.3.2, le *sysctl* `kptr_restrict` est une interface qui définit l'affichage des adresses des symboles du noyau Linux. Lorsqu'un utilisateur souhaite lire ces adresses, il lance la commande `"cat /proc/kallsyms"`. C'est la fonction `s_show` du fichier `kallsyms` qui est chargée de révéler les valeurs des pointeurs à l'aide d'une boucle sur une fonction d'affichage du type `printf`. La chaîne de caractères `"%pK %c %s\n"` est utilisée par `printf` pour afficher les adresses au bon format (voir l'ex. la Fig. 5.12, p.199 de ce chapitre). En fonction de la valeur de `kptr_restrict`, le *format specifier* `"%pK"` affiche l'adresse d'un pointeur ou un masque de zéros [118].

Dans son attaque, Alephzain cherche à modifier la valeur `"%pK"` en `"%p"` pour que la fonction d'affichage ne tienne plus compte des restrictions d'accès et laisse apparaître systématiquement l'adresse du pointeur passé en argument. Pour localiser le *format specifier* à modifier, le pirate éthique cartographie la mémoire avec la fonction `mmap`[28], puis il recherche la chaîne de caractères `"%pK %c %s\n"`. Ceci est rendu possible grâce à la vulnérabilité de lecture et d'écriture du fichier `/dev/exynos-mem`. Une fois la chaîne de caractères identifiée, celle-ci est remplacée par `"%p %c %s\n"`. La liste des pointeurs d'adresse est maintenant disponible lors du prochain appel du fichier `/proc/kallsyms`.

En utilisant les capacités de débogage proposées par le JTAG, nous allons reproduire cette démarche en recherchant directement la chaîne de caractères `"%pK %c %s\n"` dans la mémoire du système. Cependant, la taille des données à analyser n'est pas négligeable. La mémoire flash de la carte de développement a une capacité de stockage de 4Go. Analyser l'intégralité de celle-ci est envisageable mais par souci de rapidité nous nous sommes basés sur une information donnée dans le chapitre 10 de [49]. Dans cet ouvrage, les auteurs expliquent qu'une grande majorité des appareils fonctionnant sous Android utilisent une séparation mémoire à 3Go. L'espace réservé au noyau occupe les adresses supérieures à 3Go ($\geq$`0xC000_0000`) et l'espace utilisateur occupe les 3Go d'adresses restantes ($\leq$`0xC000_0000`). C'est donc à partir de cette adresse que nous allons rechercher la chaîne de caractère à modifier en effectuant un balayage vers les adresses croissantes. De la même manière que la méthode utilisée dans la Section 5.4.3 pour localiser les adresses des chaînes de caractères, la procédure que nous appliquons pour rechercher la chaîne `"%pK %c %s\n"` consiste à extraire les données de la mémoire, puis d'appliquer un algorithme de reconnaissance de forme sur celles-ci. Nous accédons directement aux valeurs stockées dans la mémoire, par conséquent, les valeurs manipulées sont des mots de 32 bits notés en valeurs hexadécimales représentant le code machine.

Une fois que la valeur `"0x204b7025 0x25206325 0x00000a73"` (`"%pK %c %s\n"`) est localisée, elle est remplacée par `"0x20207025 0x25206325 0x00000a73"` (`"%p %c %s\n"`). De cette manière, en exécutant la commande `"cat /proc/kallsyms | grep setresuid"` à travers l'interface ADB, nous avons obtenu l'adresse de la fonction système `sys_setresuid`, localisée à l'emplacement `0xC00A_2954`.

28. http://man7.org/linux/man-pages/man2/mmap.2.html

5.4.4.6.2 Injecter la faute

La manipulation précédente permet d'obtenir l'adresse de la fonction dans laquelle la perturbation doit être injectée. Avec la sonde de débogage, il est possible d'observer le code de `sys_setresuid` dans la mémoire. La figure 5.15 présente l'interface fournie par la sonde de débogage Lauterbach. L'opération de comparaison ciblée est localisée à l'adresse `0xC00A_298C`. Sur l'image de droite, on distingue que cette opération est suivie par une instruction de branchement conditionnel `"bne 0xC00A_2A14"`. Par conséquent, si `"r0 = 0x0"`, alors le branchement n'est pas fait, sinon le programme se poursuit à l' adresse `0xC00A_2A14`. Modifier la valeur de l'instruction de comparaison de `"cmp r0,#0x0"` en `"cmp r0,#0x1"` (`0xE3500000` en `0xE3500001`), impacte directement cette branche conditionnelle. Lors d'analyses faites au cours de diverses expérimentations, il a été observé que $\forall n \in \mathbb{N}^*$, `0x00`$< n \leq$`0xFF`, `"cmp r0,#"`$n \Rightarrow$ « Attribuer l'UID 0 à un processus, excepté si le demandeur a lui-même l'UID 0 ». Tant que $n \neq 0$, la fonction effectue la tâche inverse de celle pour laquelle elle a été conçue. Nous en déduisons que l'attaque réussit lorsque cette branche conditionnelle n'est pas empruntée.

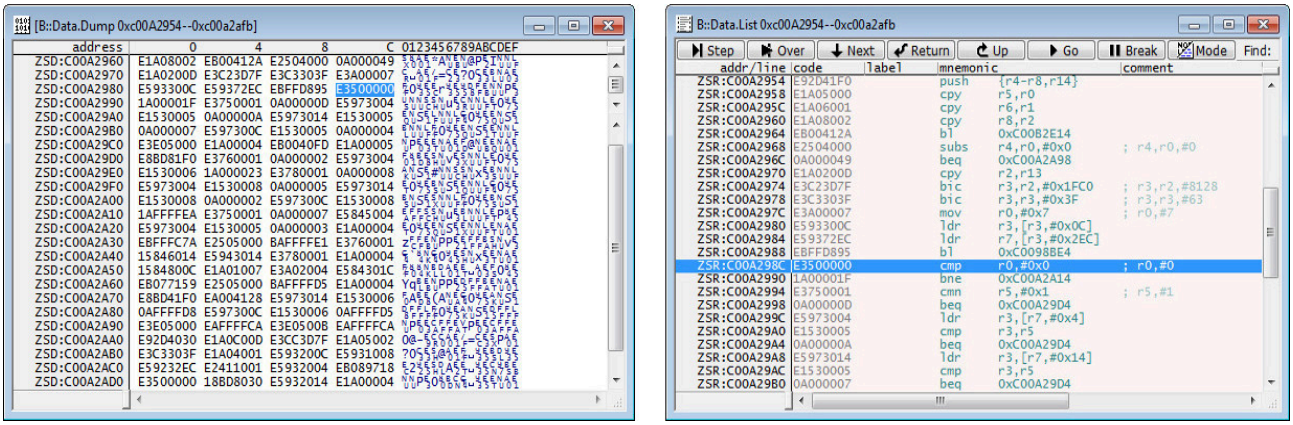


FIGURE 5.15 – Affichage de la mémoire avec la sonde de débogage Lauterbach. La figure de gauche présente le code de la fonction `sys_setresuid` sous forme brute. La figure de droite présente le même code, mais désassemblé avec une fonctionnalité proposée par cette sonde. L'opération de comparaison ciblée est surlignée en bleu.

5.4.4.6.3 Demander les privilèges `root`

Il s'agit de l'étape finale de l'attaque. Puisque la fonction `sys_setresuid` a été modifiée dans l'étape précédente, un processus non privilégié n'a qu'à demander les droits `root` pour les obtenir. Dans le cadre de notre attaque, ceci est effectué à travers l'interface d'ADB avec le programme `jtag_based_abused` dont le code source est donné Fig. 5.14, p.205 de ce chapitre. Le répertoire `/data/local/tmp` est destiné à recueillir des fichiers temporaires d'applications. Aucun privilège particulier n'est requis pour charger un programme et l'exécuter depuis ce répertoire.

5.4.4.7 Résultats expérimentaux

En exécutant le programme développé pour notre attaque avec la commande `"./jtag_based_abused"`, on observe un changement dans l'invite de commande de la fenêtre de l'interface d'ADB. En effet, le caractère de fin de ligne passe de '`$`' à '`#`'. Ceci indique qu'à partir de cet instant, toutes les commandes qui sont saisies ont les privilèges administrateur `root`. Pour s'en assurer, la commande `id` permet de connaître la valeur de notre UID. La figure 5.16 montre la fenêtre ADB à l'issue de l'exécution de toutes ces commandes.

```
user@android:/data/local/tmp $ id
uid=2000(shell) gid=2000(shell) context=u:r:shell:s0
user@android:/data/local/tmp $ ./jtag_based_abused

user@android:/data/local/tmp # id
uid=0(root) gid=2000(shell) context=u:r:shell:s0
user@android:/data/local/tmp # id
```

FIGURE 5.16 – Synthèse des commandes passées par la fenêtre de l'interface ADB

Remarquons que, maintenant que nous avons l'UID 0, il est nécessaire de rétablir la valeur initiale de la comparaison dans la fonction `sys_setresuid` avec le JTAG. Autrement, les processus lancés depuis notre `shell root` risquent de ne pas obtenir les accès qu'ils demandent puisque la fonction est inversée.

5.4.5 Utilisation du débogage matériel comme vecteur d'injection de fautes : conclusion

Ces deux applications du JTAG dans des scénarios d'attaques ont montré, qu'avec les outils de débogage, il est possible de modifier du code à la volée dans le système d'exploitation d'un SoC. En utilisant uniquement les fonctionnalités proposées par le JTAG, des valeurs impliquées dans la configuration de la sécurité du système Android ont été altérées. Ce principe peut être généralisé à n'importe quel OS tant qu'un port de débogage matériel est ouvert. Le débogage matériel peut donc être considéré comme un nouveau moyen d'injection de fautes dans les systèmes électroniques complexes.

5.5 Conclusion

Le débogage matériel est un mécanisme présent dans la plupart des systèmes sur puce. Deux aspects liés à cet outil ont été étudiés dans ce chapitre : les mécanismes de protection des accès au débogage et un nouveau type d'utilisation détournée.

Dans une première partie, Section 5.3, nous avons étudié les mécanismes de sécurité contrôlant les accès aux fonctionnalités du débogage. En sélectionnant un cas d'étude et en analysant son fonctionnement, il a été possible de contourner la sécurité du mécanisme grâce à des perturbations électromagnétiques. La mise en place de cette attaque a montré que l'application de la méthodologie proposée dans le chapitre 2, permet de localiser d'éventuels points sensibles, puis de les solliciter avec des perturbations afin de vérifier la présence d'éventuelles vulnérabilités. Cette étude a démontré qu'en l'absence de contre-mesures, il est possible d'appliquer relativement facilement des attaques matérielles sur de tels mécanismes.

La seconde partie de ce chapitre, Section 5.4, était consacrée à la présentation d'une nouvelle utilisation détournée des mécanismes de débogage. Ces derniers étaient déjà exploités pour extraire ou écraser des données en mémoire des systèmes complexes. Nous avons eu l'idée d'exploiter leurs fonctionnalités pour les utiliser en tant que vecteurs d'injection de fautes. Nous avons montré la précision des injections dans deux scénarios d'attaque. Leur succès n'est limité que par un accès libre à l'interface de débogage. Ceci pose la question de la légitimité d'étudier ces outils de débogage comme moyens de perturbation. Deux approches sont ainsi envisageables :

Accéder au débogage matériel

De nos jours, la majorité des concepteurs de SoC sont conscients du risque que représente un accès libre au débogage matériel et, par conséquent, proposent des solutions de sécurité. Comme expliqué dans la littérature académique telle que [119, 19], il est possible de prendre des mesures préventives pour assurer une intégrité aux mécanismes de débogage. Il est également envisageable de sécuriser la chaîne de *Boundary-Scan* des composants comme le proposent les travaux [107] et [18]. Plus récemment, les auteurs de [115] proposent un mécanisme pour analyser les actions de débogage effectuées afin de détecter des comportements suspects et prendre les mesures en conséquence. Cependant, malgré l'existence de toutes ces solutions innovantes, la majorité des systèmes présents sur le marché actuel n'utilisent que des sécurités au niveau de l'accès au débogage comme dans [126] ou dans le système étudié Section 5.3. Or, nous avons montré qu'une implémentation de cette sécurité sans contre-mesure, pouvait être cassée au moyen d'attaques matérielles. Une fois le mécanisme de sécurité forcé, il est légitime de considérer le débogage matériel comme un moyen d'injection de fautes.

Utiliser le débogage pour évaluer la faisabilité et l'efficacité de l'injection d'une faute

Sans chercher à casser le mécanisme de verrouillage du débogage matériel, il est envisageable d'utiliser ce dernier pour évaluer la faisabilité d'une attaque par injection de faute sur un système donné. En se procurant une carte de développement intégrant un

SoC similaire à celui dont le débogage est verrouillé, un attaquant peut simuler la faisabilité d'une attaque. Les deux applications d'attaques exploitant le JTAG dans la section 5.4 ont été accomplies en ne modifiant que la valeur d'un unique bit. Certes, de grandes contraintes au niveau de la précision de l'injection des perturbations sont présentes mais nous avons montré dans le Ch. 3 qu'en appliquant la méthodologie proposée dans le Ch. 2, il est possible de perturber une opération cryptographique précise. Par conséquent, il est envisageable de penser que les perturbations effectuées grâce au JTAG sont également réalisables en utilisant les champs électromagnétiques.

Ces deux approches montrent que le débogage matériel peut être considéré comme un moyen de perturbation à part entière, ou comme un outil d'aide à la conception d'attaques matérielles. Dans les deux cas, il est primordial de limiter son accès uniquement aux utilisateurs désignés, avec des mécanismes de protection adaptés aux différentes menaces.

Conclusion générale et perspectives

Tout au long de ce manuscrit, nous avons démontré l'applicabilité des attaques matérielles sur des systèmes complexes embarqués. Notre travail souligne le fait que ce n'est pas tant le coût ni la complexité du matériel utilisé pour mener des attaques physiques qui augmente avec celle du système ciblé, mais plutôt la complexité de la méthode et du traitement des données. Dans cette étude, nous montrons qu'en utilisant une grandeur physique appropriée et une méthode de test adaptée, il est possible d'appréhender la complexité des systèmes sur puce pour perpétrer des attaques matérielles. Les systèmes que nous avons considérés sont pour la plupart, soumis à la propriété intellectuelle, ce qui signifie une absence d'information publique précise au sujet de leur fonctionnement. Néanmoins, nous avons mis en évidence que lorsqu'un système de sécurité n'implémente pas de contre-mesure le protégeant des attaques physiques, il est possible de détecter des vulnérabilités et, dès lors, de forcer la sécurité. Par conséquent, la complexité des systèmes sur puce ne les protège pas contre les attaques matérielles. C'est pourquoi, les mécanismes de sécurité intégrés dans les SoC doivent être implémentés en considérant tous les types de menaces, et en particulier les attaques matérielles.

Résumé des contributions

Dans le chapitre 2 nous présentons la mise en œuvre du contexte expérimental nécessaire à cette étude. La méthodologie appliquée dans les différentes manipulations de ce document y est explicitée. Elle a pour objectif d'établir un schéma de test générique applicable à n'importe quel système de sécurité évalué face à des attaques matérielles. Dans ce chapitre, nous procédons également à une réflexion sur les chemins d'attaques et les grandeurs physiques exploitables dans les SoC. Il en résulte l'utilisation des champs électromagnétiques avec un guide destiné à faciliter la configuration de l'ensemble des paramètres liés à cette grandeur. De plus, une présentation des bancs de test que nous avons dû adapter pour nos travaux y est détaillée.

Les chapitres 3 et 4 exposent l'application d'attaques matérielles utilisant les champs électromagnétiques sur des modules de sécurité intégrés dans les SoC. Dans le chapitre 3, des attaques par canaux cachés et par injections de fautes sont effectuées contre des implémentations cryptographiques. En comparant des attaques tantôt sur une implémentation logicielle, tantôt sur une implémentation matérielle du même algorithme de chiffrement, nous avons souligné l'importance de la méthodologie à considérer en fonction de la cible. L'implémentation logicielle sans contre-mesure s'est révélée fragile et relativement simple à attaquer. En revanche l'implémentation matérielle qui consistait en un crypto-accélérateur

protégé contre les DPA, a nécessité une multitude d'expérimentations et le développement d'outils statistiques afin de dévoiler d'éventuelles vulnérabilités. Ceci a mis en avant le fait que, devant la quantité d'informations à traiter, dans le pire des cas, il est essentiel d'automatiser et d'asservir le plus de paramètres possible afin d'optimiser le temps de l'évaluation. D'autre part, tester un maximum de paramètres de manière méthodique permet, lorsqu'aucune vulnérabilité n'est identifiée, de définir une borne inférieure qui quantifie le nombre d'expérimentations ne dévoilant pas de faille. Dans le cas d'une détection de faiblesse, cela permet de conclure quant à la robustesse du système.

Le chapitre 4 présente l'exploitation des champs magnétiques pour perturber les systèmes de sécurité des environnements d'exécution de confiance. Cette étude a prouvé qu'il est possible d'attaquer des OS complexes, en détectant de manière méthodique les éventuelles vulnérabilités qui n'ont pas été corrigées. De plus, le fait de considérer des implémentations propriétaires, sans informations autres que celles disponibles pour les développeurs, montre que la rétention d'informations ne fait que retarder l'instant où les implémentations vulnérables aux attaques matérielles tombent face à celles-ci. Ceci met en évidence qu'une évaluation en « boîte noire » d'un système de protection ne représente pas le niveau réel de sa sécurité, mais plutôt une estimation du temps nécessaire à un attaquant pour détecter une éventuelle vulnérabilité.

Enfin, dans le chapitre 5, les systèmes de débogage matériels ont été considérés comme nouveaux chemins d'attaques. Dans une première partie, la méthodologie de test développée dans cette étude a été appliquée avec les champs EM sur un mécanisme de verrouillage d'accès au débogage. Cette démarche a confirmé les observations précédentes : il est possible de contourner une sécurité non conçue pour résister aux attaques matérielles. D'autre part, dans ce chapitre, nous présentons le débogage comme un nouveau moyen d'injection de faute dans un système complexe. Cet outil très utile aux développeurs, facilite également la tâche aux attaquants pour appréhender la complexité de ces systèmes.

Dans son ensemble, ce travail fournit une base décrivant les possibilités de mise en œuvre d'attaques matérielles sur des systèmes complexes embarqués. Par conséquent, les concepteurs et les développeurs peuvent s'appuyer sur celui-ci afin d'élargir le spectre de la conception des menaces qui guettent leurs implémentations et élaborer des solutions mieux adaptées. Par exemple, comme nous l'avons montré dans cette étude, un test conditionnel lié à une valeur impliquée dans une sécurité quelconque ne doit jamais être effectué avec un simple "`if`". En effet, un attaquant pourrait perturber ce test de manière matérielle et détourner l'exécution du code. Pour parer à cette éventualité, l'ajout de vérification du flot d'exécution ou une duplication du "`if`" conviendrait. Ainsi, avec la connaissance de ce type d'attaques, le schéma et la structure des solutions de sécurité choisies seront mieux adaptées au contexte dans lequel sera placé le produit final. Ce travail pourrait ainsi inspirer la rédaction d'un recueil des « bonnes pratiques » dans l'implémentation des mécanismes de sécurité, qui conseillerait sur la nature des contre-mesures à appliquer en fonction des situations.

Perspectives et améliorations

Dans cette étude, nous avons volontairement choisi de restreindre nos expérimentations aux attaques matérielles *non invasives*. Ceci nous a amené à considérer les champs élec-

tromagnétiques comme étant la grandeur physique la plus adaptée dans le contexte des systèmes complexes embarqués. Cette grandeur a la particularité d'être exploitable pour l'ensemble des attaques matérielles à savoir, rétroconception, attaque par canaux cachés et par injections de faute. Elle s'utilise au travers des boîtiers et offre l'avantage d'avoir un champ d'application localisé. Cependant, les paramètres liés à cette grandeur représentent un problème complexe d'optimisation. Si nous avons proposé un guide afin d'établir ces derniers, notre méthode n'en reste pas moins expérimentale. En effet, plusieurs études sont à développer afin d'optimiser l'utilisation de cette grandeur physique.

Une première démarche consisterait à étudier les relations entre les caractéristiques physiques des injecteurs, les champs générés et les fautes produites en fonction de la technologie de fabrication des systèmes attaqués. Cette recherche se déroulerait en association avec la communauté de la compatibilité électromagnétique, afin de concevoir un modèle décrivant les dépendances entre l'impulsion envoyée par le générateur électrique et les spécificités du champ EM produit. Ceci permettrait de comprendre plus précisément les interactions avec les zones impactées du système. De plus, cette étude proposerait une calibration et un étalonnage des injecteurs afin d'établir une méthode de sélection de ces derniers plus formelle lors de tests d'évaluation.

Une autre exploration orientée sur la dualité observations-injections électromagnétiques permettrait d'optimiser le nombre de manipulations entre la détection de fuites d'informations d'un processus donné et la perturbation de celui-ci. Les sondes de mesures et les injecteurs que nous avons utilisés possèdent des facteurs de formes différents (voir Ch. 2). Il serait intéressant d'étudier le développement d'un dispositif capable de mesurer et de générer des impulsions EM. Cette sonde-injecteur aurait le même facteur de forme et permettrait d'établir une relation plus fine entre les émissions EM et les zones perturbées.

D'autre part, nous avons vu dans cette étude que la complexité des SoC demandait l'acquisition d'un plus grand nombre de données : cartographies EM, nombre de mesures, etc. Pour cela, nous avons adapté des bancs de tests déjà existants. Cependant, afin d'optimiser de futures expérimentations, il serait intéressant de concevoir des outils spécifiques, correctement dimensionnés pour de tels systèmes. Ces outils seraient capables d'acquérir, stocker et traiter un grand nombre de données sans faire exploser les temps de manipulation. Ceci permettrait d'augmenter le nombre de traces pour tenter d'améliorer le RSB des mesures. Une certaine connexité avec la problématique du « big data » peut être relevée et d'éventuelles solutions de ce domaine pourraient être appliquées ici, par exemple l'utilisation d'algorithmes distribués, de statistiques inférentielles, etc.

Nous avons également souligné la nécessité d'asservir et d'automatiser l'ajustement des paramètres des cartographies de mesures et injections EM. Pour l'aspect « side-channel », le développement de méthodes adaptées pour révéler les fuites d'informations (comme la TVLA [27] par exemple) doit être étudiée. Pour l'aspect injection de fautes, les paramètres de localisation et ceux de la grandeur physique utilisée (intensité, durée, etc.) doivent être ajustés selon un processus optimisé. Prenons l'exemple de l'ajustement de l'amplitude d'une perturbation EM. Considérons un scénario de test dans lequel une cartographie d'injections EM est effectuée sur la surface d'un SoC. Une perturbation doit être répétée 50 fois au-dessus de chaque point spatial de cette cartographie. Si, au dessus d'un point donné, une perturbation ne produit que des redémarrages du système, il est pertinent d'établir un processus qui recherche la valeur de l'amplitude de l'impulsion qui amènera ce

système à la limite de son « plantage » afin de produire d'éventuelles fautes. L'étude des méthodes d'ajustements serait intéressante afin de gagner un temps considérable dans les manipulations. L'apprentissage automatique dans ce domaine est tout indiqué.

Enfin, nous avons montré dans le dernier chapitre, que le débogage matériel était un vecteur d'attaque pertinent. Cependant, les sondes de débogage que nous avons utilisées étaient des outils commerciaux qui n'étaient pas destinés à évaluer de possibles attaques liées aux débogages. Dans [95], les auteurs ont conçu une sonde basique, instrumentable, afin de tester la robustesse d'un mécanisme de verrouillage. Il serait intéressant de s'inspirer de ce principe et de développer une sonde non pas destinée à déboguer un système, mais conçue pour envoyer un minimum de séquences de test afin d'évaluer la robustesse des systèmes de sécurité de débogage. Cet outil serait totalement paramétrable et aurait des caractéristiques de fonctionnement capables de pousser les contrôleurs de débogage au-delà de leur mode de fonctionnement nominal. On pourrait également imaginer des fonctionnalités supplémentaires, telles que le « fuzzing » dans les instructions, pour chercher à détecter des comportements imprévus. Cette sonde serait totalement instrumentable et intégrée dans les bancs de tests destinés aux systèmes complexes.

Communications et présentations

Publication dans un journal international

1. F. Majéric, B. Gonzalvo, and L. Bossuet. Jtag fault injection attack. *IEEE Embedded Systems Letters*, 10(3) :65-68, Sept 2018.

Publications dans des conférences internationales

2. F. Majéric, E. Bourbao, and L. Bossuet. Electromagnetic security tests for soc. In *2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 265–268, Dec 2016.

3. F. Majéric, B. Gonzalvo, and L. Bossuet. Jtag combined attack - another approach for fault injection. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, Nov 2016.

Communications non actées

4. F. Majéric. Technologie ARM TrustZone et attaques combinées sur SoC. *GDR-SoC2 – Sécurité des SoC complexes hétérogènes – de la TEE au matériel*, Sept 2018, Campus de Jussieu, Paris.

5. F. Majéric. EM injections on cryptographic implementations on SoC. *Journée thématique – Injection de fautes : attaques physiques, protections logicielles et mécanismes d'évaluation de la robustesse*, Mai 2018, Campus de Jussieu, Paris.

6. F. Majéric, E. Bourbao, and L. Bossuet. Reversing the field to attack the SoCs – Double use of EM-fields to defeat the complexity. *Conference on trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE)*, Nov 2016, Barcelone, Spain.

7. F. Majéric, E. Bourbao, and L. Bossuet. Double use of EM-fields to defeat the SoCs complexity. *Workshop in Practical Hardware Innovation in Security and Characterisation (PHISIC)*, Oct 2016, Gardanne, France.

8. F. Majéric, E. Bourbao, and L. Bossuet. Reversing the Field : a Single Channel for Self-Combined Attack (to Defeat Circuit Complexity). *International CryptArchi Workshop*, Jun 2016, La Grande Motte, France.

9. F. Majéric, B. Gonzalvo, and L. Bossuet. Jtag Combined Attack. *Conference on trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE)*, Sep 2015, Saint-Malo, France.

Bibliographie

[1] Forensic analysis of mobile phone internal memory. In Mark Pollitt and Sujeet Shenoi, editors, *Advances in Digital Forensics*, volume 194 of *IFIP — The International Federation for Information Processing*. (Cité pages 26, 32, 175 et 194.)

[2] Privilege escalation attacks on android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ilić, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*. (Cité page 202.)

[3] Federal Information Processing Standards Publication 197 : Advanced Encryption Standard, Nov 2001. [Online] Available : https://doi.org/10.6028/NIST.FIPS.197. (Cité pages 14, 59 et 61.)

[4] *Embedded Cryptographic Hardware : Methodologies & Architectures*. Nova Science Publishers Inc., october 2004. (Cité page 31.)

[5] JEDEC design Std. Fine-pitch, Square Ball Grid Array Package (FBGA) Package-on-Package (PoP) – JEDEC solid state technology association, October 2008. Committee(s) JC-11 JC-11.2, Publication 95, Design guide 4.22. (Cité page 237.)

[6] IEEE Std 1149.1-2013 : IEEE Standard for Test Access Port and Boundary-Scan Architecture – IEEE Computer Society, May 2013. (Revision of IEEE Std 1149.1-2001). (Cité pages 177, 269 et 270.)

[7] JEDEC Design Requirements - Ball Grid Array Package (BGA) – JEDEC solid state technology association, March 2015. Committee(s) JC-11.2, Publication 95, Design guide 4.14. (Cité page 236.)

[8] Josh Aas. Understanding the linux 2.6. 8.1 cpu scheduler. *Retrieved Oct*, 16 :1–38, 2005. (Cité page 75.)

[9] D. Agrawal, B. Archambeault, J.R. Rao, and P. Rohatgi. The em side-channel(s). In *CHES*, pages 29–45. LCNS 2523, 2003. (Cité page 42.)

[10] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. *The EM Side—Channel(s)*, pages 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. (Cité page xxxvii.)

[11] Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, and Didier Moyart. *Power Analysis, What Is Now Possible...*, pages 489–502. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. (Cité page xxxvii.)

[12] Alephzain. xda-developers > samsung galaxy s iii i9300, i9305 > galaxy s iii general > [root][security] root exploit on exynos. [Online] Available : http://forum.xda-developers.com/galaxy-s3/general/root-root-exploit-exynos-t2047991, December 2012. (Cité page 203.)

[13] F. Amiel, K. Villegas, B. Feix, and L. Marcel. Passive and active combined attacks : Combining fault attacks and side channel analysis. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pages 92–102, Sept 2007. (Cité page xxxvii.)

[14] Stéphanie Anceau, Pierre Bleuet, Jessy Clédière, Laurent Maingault, Jean-luc Rainard, and Rémi Tucoulou. Nanofocused x-ray beam to reprogram secure circuits. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 175–188, Cham, 2017. Springer International Publishing. (Cité page 24.)

[15] Apple. Sécurité ios - ios11. [Online] Available : https://images.apple.com/fr/business/docs/iOS_Security_Guide.pdf, janvier 2018. (Cité page 145.)

[16] ARM. Amba. http://infocenter.arm.com/help/topic/com.arm.doc.set.amba, 2013. (Cité page 250.)

[17] ARM. Trustzone. [Online] Available : https://www.arm.com/products/security-on-arm/trustzone/tee-reference-documentation, 2018. (Cité page 134.)

[18] J. Backer, D. Hély, and R. Karri. Secure design-for-debug for systems-on-chip. In *2015 IEEE International Test Conference (ITC)*, pages 1–8, Oct 2015. (Cité page 210.)

[19] Jerry Backer, David Hely, and Ramesh Karri. Secure and flexible trace-based debugging of systems-on-chip. *ACM Trans. Des. Autom. Electron. Syst.*, 22(2) :31 :1–31 :25, December 2016. (Cité page 210.)

[20] J. Balasch, B. Gierlichs, and I. Verbauwhede. An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 105–114, Sept 2011. (Cité page 13.)

[21] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2) :370–382, Feb 2006. (Cité pages xxxvii, 24 et 25.)

[22] Guillaume Barbu, Guillaume Duc, and Philippe Hoogvorst. *Java Card Operand Stack : Fault Attacks, Combined Attacks and Countermeasures*, pages 297–313. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. (Cité page 13.)

[23] Guillaume Barbu, Hugues Thiebeauld, and Vincent Guerin. *Attacks on Java Card 3.0 Combining Fault and Logical Attacks*, pages 148–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. (Cité page 16.)

[24] Pierre Bayon. *Attaques électromagnétiques ciblant les générateurs d'aléa.* PhD thesis, 2014. Thèse de doctorat dirigée par Bossuet, Lilian et Aubert, Alain Microélectronique Saint Etienne 2014. (Cité page 42.)

[25] Pierre Bayon, Lilian Bossuet, Alain Aubert, Viktor Fischer, François Poucheret, Bruno Robisson, and Philippe Maurine. *Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator*, pages 151–166. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. (Cité page 22.)

[26] Friedrich Beck. *Integrated Circuit Failure Analysis : A Guide to Preparation Techniques.* Wiley, January 1998. (Cité pages 10 et 268.)

[27] G Becker, J Cooper, E DeMulder, G Goodwill, J Jaffe, G Kenworthy, T Kouzminov, A Leiserson, M Marson, P Rohatgi, et al. Test vector leakage assessment (tvla) methodology in practice. In *International Cryptographic Module Conference*, volume 1001, page 13, 2013. (Cité pages 79 et 215.)

[28] Noemie Beringuier-Boher, Marc Lacruche, David El-Baze, Jean-Max Dutertre, Jean-Baptiste Rigaud, and Philippe Maurine. Body biasing injection attacks in practice. In *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, CS2 '16, pages 49–54, New York, NY, USA, 2016. ACM. (Cité page 20.)

[29] Eli Biham and Shamir Adi. *Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag New York, 1993. (Cité page xxxvii.)

[30] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 513–525, London, UK, UK, 1997. Springer-Verlag. (Cité pages xxxvii et 14.)

[31] Johannes Blömer and Jean-Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (aes). In Rebecca N. Wright, editor, *Financial Cryptography*, pages 162–181, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cité page 72.)

[32] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults*, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. (Cité page 13.)

[33] O. Breitenstein, F. Altmann, T. Riediger, D. Karg, and V. Gottschalk. Lock-in thermal ir imaging using a solid immersion lens. *Microelectronics Reliability*, 46(9) :1508 – 1513, 2006. Proceedings of the 17th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis. Wuppertal, Germany 3rd–6th October 2006. (Cité page 25.)

[34] Eric Brier, Christophe Clavier, and Francis Olivier. *Correlation Power Analysis with a Leakage Model*, pages 16–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. (Cité pages xxxvii, 71, 84, 256 et 260.)

[35] Dominik Brodowski. Cpu frequency and voltage scaling code in the linux (tm) kernel. *Linux kernel documentation*, 2013. (Cité page 66.)

[36] J. A. Butts and G. S. Sohi. A static power model for architects. In *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, pages 191–201, 2000. (Cité pages 14 et 249.)

[37] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cité page 83.)

[38] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. *Template Attacks*, pages 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. (Cité page 262.)

[39] Chien-Ning Chen and Sung-Ming Yen. Differential fault analysis on aes key schedule and some countermeasures. In Rei Safavi-Naini and Jennifer Seberry, editors, *Information Security and Privacy*, pages 118–129, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cité page 72.)

[40] Patrick Colp, Jiawen Zhang, James Gleeson, Sahil Suneja, Eyal de Lara, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Protecting data on smartphones and tablets from memory attacks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 177–189, New York, NY, USA, 2015. ACM. (Cité page 25.)

[41] ARM CoreSight Components. Technical reference manual. ARM DDI 0314H. Copyright © 2004-2009 ARM. All rights reserved. (Cité pages 178, 179, 275 et 276.)

[42] Jean-Sébasticn Coron, Paul Kocher, and David Naccache. Statistics and secret leakage. In Yair Frankel, editor, *Financial Cryptography*, pages 157–173, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. (Cité page 83.)

[43] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3) :273–297, 1995. (Cité page 262.)

[44] ARM Cortex-A9. Technical reference manual. ARM 100511_0401_10_en. Copyright © 2008-2012, 2016 ARM. All rights reserved. (Cité page 28.)

[45] Amitabh Das, Jean Da Rolt, Santosh Ghosh, Stefaan Seys, Sophie Dupuis, Giorgio Di Natale, Marie-Lise Flottes, Bruno Rouzeyre, and Ingrid Verbauwhede. Secure jtag implementation using schnorr protocol. *Journal of Electronic Testing*, 29(2) :193–209, Apr 2013. (Cité page 182.)

[46] Elke De Mulder. *: Electromagnetic Techniques and Probes for Side-Channel Analysis on Cryptographic Devices*. PhD thesis, Nov 2010. (Cité page 21.)

[47] A. Dehbaoui, J.M. Dutertre, B. Robisson, P. Orsatelli, P. Maurine, and A. Tria. Injection of transient faults using electromagnetic pulses -practical results on a cryptographic system-. Cryptology ePrint Archive, Report 2012/123, 2012. http://eprint.iacr.org/2012/123. (Cité page 22.)

[48] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1(2) :123, 2011. (Cité page 256.)

[49] Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva Fora, Stephen A. Ridley, and Georg Wicherski. *Android Hacker's Handbook*. Wiley Publishing, 1st edition, 2014. (Cité page 207.)

[50] Pierre Dusart, Gilles Letourneux, and Olivier Vivolo. Differential fault analysis on a.e.s. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *Applied Cryptography and Network Security*, pages 293–306, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cité pages 72, 73, 76, 92 et 109.)

[51] R. Elbaz, L. Torres, G. Sassatelli, P. Guillemin, C. Anguille, C. Buatois, and J. B. Rigaud. Hardware engines for bus encryption : a survey of existing techniques. In *Design, Automation and Test in Europe*, pages 40–45 Vol. 3, March 2005. (Cité page 31.)

[52] J. Ferrigno and M. Hlaváč. When aes blinks : introducing optical side channel. *IET Information Security*, 2 :94–98), September 2008. (Cité pages 23 et 265.)

[53] Michael J. Flynn and Wayne Luk. *Computer System Design - System-on-Chip*. Wiley, 1st edition, october 2011. (Cité pages 233, 235 et 251.)

[54] Pascal Fouillat. *Contribution à l'étude de l'interaction entre un faisceau laser et un milieu semiconducteur : applications à l'étude du latchup et à l'analyse d'états logiques dans les circuits intégrés en technologie CMOS*. PhD thesis, 1990. Thèse de doctorat dirigée par Dom, Jean-Paul. (Cité page 24.)

[55] Hiroshi Fuketa, Masanori Hashimoto, Yukio Mitsuyama, and Takao Onoye. Trade-off analysis between timing error rate and power dissipation for adaptive speed control with timing error prediction. In *2009 Asia and South Pacific Design Automation Conference*, pages 266–271, Jan 2009. (Cité page 250.)

[56] Priyanka Gandhani. Moving from amba ahb to axi bus in soc designs : A comparative study. *International Journal of Computer Science & Emerging Technologies*, 2(4), 2011. (Cité page 250.)

[57] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis : Concrete results. In *CHES*, pages 251–261. LCNS 2162, 2001. (Cité pages 21 et 42.)

[58] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. Ecdsa key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1626–1638, New York, NY, USA, 2016. ACM. (Cité page 59.)

[59] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and*

Embedded Systems - CHES 2006, pages 15–29, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. (Cité pages 80 et 83.)

[60] Christophe Giraud. *DFA on AES*, pages 27–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. (Cité pages 14 et 72.)

[61] GlobalPlatform. Trusted execution environment (tee). [Online] Available : https://www.globalplatform.org/specificationsdevice.asp, 2018. (Cité page 146.)

[62] D. H. Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5) :91–100, Oct 1965. (Cité page 24.)

[63] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember : cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5) :91–98, 2009. (Cité page 25.)

[64] Helena Handschuh, Pascal Paillier, and Jacques Stern. Probing attacks on tamper-resistant devices. In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, pages 303–315, London, UK, UK, 1999. Springer-Verlag. (Cité page 19.)

[65] D. Hely, M. L. Flottes, F. Bancel, B. Rouzeyre, N. Berard, and M. Renovell. Scan design and secure chip [secure ic testing]. In *Proceedings. 10th IEEE International On-Line Testing Symposium*, pages 219–224, July 2004. (Cité page 181.)

[66] Nadia Heninger. *Cold-Boot Attacks*, pages 216–217. Springer US, Boston, MA, 2011. (Cité page 25.)

[67] William Hohl and Christopher Hinds. *ARM Assembly Language : Fundamentals and Techniques, Second Edition*. CRC Press, 2nd edition, February. (Cité page 237.)

[68] Mirko Holler, Manuel Guizar-Sicairos, Esther H. R. Tsai, Roberto Dinapoli, Elisabeth Müller, Oliver Bunk, Jörg Raabe, and Gabriel Aeppli. High-resolution non-destructive three-dimensional imaging of integrated circuits. In *Nature*, volume 543, pages 402–406. Macmillan Publishers Limited, part of Springer Nature. All rights reserved, March 2017. http://dx.doi.org/10.1038/nature21698. (Cité pages 265 et 266.)

[69] Andrew Huang. Keeping secrets in hardware : The microsoft xboxtm case study. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 213–227, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cité page 177.)

[70] ARM Debug Interface. Architecture specification adiv5.0 to adiv5.2. ARM IHI 0031C (ID080813). Copyright © 2006, 2009, 2012, 2013 ARM Limited or its affiliates. All rights reserved. (Cité pages 184, 269 et 277.)

[71] P. Israsena. Design and implementation of low power hardware encryption for low cost secure rfid using tea. In *2005 5th International Conference on Information Communications Signal Processing*, pages 1402–1406, 2005. (Cité page 31.)

[72] D. Jacquet, F. Hasbani, P. Flatresse, R. Wilson, F. Arnaud, G. Cesana, T. Di Gilio, C. Lecocq, T. Roy, A. Chhabra, C. Grover, O. Minez, J. Uginet, G. Durieu, C. Adobati, D. Casalotto, F. Nyer, P. Menut, A. Cathelin, I. Vongsavady, and P. Magarshack. A 3 ghz dual core processor arm cortex tm -a9 in 28 nm utbb fd-soi cmos with ultra-wide voltage range and energy efficiency optimization. *IEEE Journal of Solid-State Circuits*, 49(4) :812–826, April 2014. (Cité page 249.)

[73] Nikhil Jayakumar, Suganth Paul, Rajesh Garg, Kanupriya Gulati, and Sunil P. Khatri. *Optimum Reverse Body Biasing for Leakage Minimization*, pages 91–100. Springer US, Boston, MA, 2010. (Cité page 249.)

[74] Lau JH. Recent advances and new trends in flip chip technology. In *ASME. Journal of Electronic Packaging*, June 2016. 138(3) :030802-030802-23, doi : 10.1115/1.4034037. (Cité page 236.)

[75] C. H. Kim and J. J. Quisquater. Faults, injection methods, and fault attacks. *IEEE Design Test of Computers*, 24(6) :544–545, Nov 2007. (Cité page 19.)

[76] Kenjiro Kimura, Kei Kobayashi, Kazumi Matsushige, Koji Usuda, and Hirofumi Yamada. Noncontact-mode scanning capacitance force microscopy towards quantitative two-dimensional carrier profiling on semiconductor devices. *Applied Physics Letters*, 90(8) :083101, 2007. (Cité page 268.)

[77] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 753–760, New York, NY, USA, 2004. ACM. Moderator-Ravi, Srivaths. (Cité page 27.)

[78] Paul C. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*, pages 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. (Cité pages xxxvii, 14, 23, 256 et 257.)

[79] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag. (Cité pages xxxvii, 18, 256 et 258.)

[80] TS Rajesh Kumar. *On-chip memory architecture exploration of embedded system on chip*. PhD thesis, Indian Institute of Science Bangalore, 2008. (Cité page 29.)

[81] K. Lahiri, S. Dey, and A. Raghunathan. *Multiprocessor Systemson-Chips*, chapter "Design of communication architectures for high-performance and energy-efficient systems-on-chip", pages 187–222. Elsevier, 2004. (Cité pages 233 et 250.)

[82] Eun-Sung Lee, Sang-Mock Lee, Daniel J. Shanefield, and W. Roger Cannon. Enhanced thermal conductivity of polymer matrix composite via high solids loading of aluminum nitride in epoxy resin. *Journal of the American Ceramic Society*, 91(4) :1169–1174, 2008. (Cité page 235.)

[83] O. Ligor, B. Gautier, A. Descamps-Mandine, D. Albertini, N. Baboux, and L. Militaru. Interpretation of scanning capacitance microscopy for thin oxides characterization. *Thin Solid Films*, 517(24) :6721 – 6725, 2009. (Cité page 268.)

[84] Null Linda and Lobur Julia. *Essentials of Computer Organization and Architecture*, chapter "Memory". Jones and Bartlett Publishers, Inc, Sudbury, United States, 4th edition, April 2015. (Cité pages 242 et 244.)

[85] J. Longo, E. De Mulder, D. Page, and M. Tunstall. Soc it to em : electromagnetic side-channel attacks on a complex system-on-chip. Cryptology ePrint Archive, Report 2015/561, 2015. http://eprint.iacr.org/2015/561. (Cité pages 59, 79 et 83.)

[86] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. *Breaking Cryptographic Implementations Using Deep Learning Techniques*, pages 3–26. Springer International Publishing, Cham, 2016. (Cité page 262.)

[87] P. Maistri, R. Leveugle, L. Bossuet, A. Aubert, V. Fischer, B. Robisson, N. Moro, P. Maurine, J. M. Dutertre, and M. Lisart. Electromagnetic analysis and fault injection onto secure circuits. In *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, Oct 2014. (Cité pages xxxvii et 42.)

[88] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*, volume 1. Springer US, 2007. (Cité page 18.)

[89] ARM® Architecture Reference Manual. Armv7-a and armv7-r edition. ARM DDI 0406C.d. Copyright © 1996-1998, 2000, 2004-2012, 2014, 2018 ARM Limited. All rights reserved. (Cité page 120.)

[90] Nobuyuki Matsuki, Ryoichi Ishihara, Alessando Baiano, and Kees Beenakker. Investigation of local electrical properties of coincidence-site-lattice boundaries in location-controlled silicon islands using scanning capacitance microscopy. *Applied Physics Letters*, 93(6) :062102, 2008. (Cité page 268.)

[91] Philippe Maurine, Karim Tobich, Thomas Ordas, and Pierre Yvan Liardet. Yet Another Fault Injection Technique : by Forward Body Biasing Injection. In *YACC'2012 : Yet Another Conference on Cryptography*, Porquerolles Island, France, September 2012. (Cité page 20.)

[92] Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against aes cryptosystem. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 91–100, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. (Cité page 72.)

[93] Tammy Noergaard. *Embedded Systems Architecture*, chapter "Board Memory". Elsvier, 2012. (Cité page 242.)

[94] NXP. i.mx 6solo/6duallite applications processor reference manual. Document Number : IMX6SDLRM - Rev. 2, April 2015. (Cité page 248.)

[95] Johannes Obermaier and Stefan Tatschner. Shedding too much light on a microcontroller's firmware protection. In *11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association*, 2017. (Cité page 216.)

[96] Jeong Wook Oh. Reverse engineering flash memory for fun and benefit. In *Blackhat USA 2014*, August. [Online] Available : https://www.blackhat.com/us-14/archives.html#Oh. (Cité page 117.)

[97] R. Omarouayache, J. Raoult, S. Jarrix, L. Chusseau, and P. Maurine. Magnetic microprobe design for em fault attack. In *2013 International Symposium on Electromagnetic Compatibility*, pages 949–954, Sept 2013. (Cité pages 47 et 54.)

[98] S. Ordas, L. Guillaume-Sage, and P. Maurine. Em injection : Fault model and locality. In *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 3–13, Sept 2015. (Cité pages 22 et 54.)

[99] S. Ordas, L. Guillaume-Sage, and P. Maurine. Electromagnetic fault injection : the curse of flip-flops. *Journal of Cryptographic Engineering*, 7(3) :183–197, Sep 2017. (Cité pages 144 et 174.)

[100] Sébastien Ordas. *Évaluation de méthodes faible consommation contre les attaques matérielles*. PhD thesis, 2015. Thèse de doctorat dirigée par Maurine, Philippe Systèmes automatiques et microélectroniques Montpellier 2015. (Cité pages 19 et 42.)

[101] Sébastien Ordas, Ludovic Guillaume-Sage, Karim Tobich, Jean-Max Dutertre, and Philippe Maurine. Evidence of a larger EM-induced fault model. In *CARDIS : Smart Card Research and Advanced Application*, volume LNCS of *Smart Card Research and Advanced Applications*, pages 245–259, Paris, France, November 2014. (Cité page 22.)

[102] Thomas Ordas, Mathieu Lisart, Etienne Sicard, Philippe Maurine, and Lionel Torres. *Near-Field Mapping System to Scan in Time Domain the Magnetic Emissions of Integrated Circuits*, pages 229–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. (Cité page 50.)

[103] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5) :879–899, May 2008. (Cité page 253.)

[104] Keun-Young Park, Sang Guun Yoo, and Juho Kim. Debug port protection mechanism for secure embedded devices. 12, 06 2012. (Cité page 181.)

[105] Kenneth P. Parker. *The Boundary-Scan Handbook*. Springer US, third edition, 2003. http://dx.doi.org/10.1007/978-1-4615-0367-5. (Cité page 270.)

[106] S. J. Pennycook, A. R. Lupini, M. Varela, A. Borisevich, Y. Peng, M. P. Oxley, K. Van Benthem, and M. F. Chisholm. *Scanning Transmission Electron Microscopy for Nanostructure Characterization*, pages 152–191. Springer New York, New York, NY, 2007. (Cité page 267.)

[107] L. Pierce and S. Tragoudas. Multi-level secure jtag architecture. In *2011 IEEE 17th International On-Line Testing Symposium*, pages 208–209, July 2011. (Cité page 210.)

[108] Gilles Piret and Jean-Jacques Quisquater. *A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad*, pages 77–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. (Cité page 14.)

[109] James D. Plummer, Michael Deal, and Peter D. Griffin. *Silicon VLSI Technology : Fundamentals, Practice, and Modeling*, chapter "Backend Technology". Pearson, 2001. (Cité page 251.)

[110] F. Poucheret, K. Tobich, M. Lisarty, L. Chusseauz, B. Robissonx, and P. Maurine. Local and direct em injection of power into cmos integrated circuits. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 100–104, Sept 2011. (Cité page 23.)

[111] Vincent Pouget. *Simulation expérimentale par impulsions laser ultra-courtes des effets des radiations ionisantes sur les circuits intégrés.* PhD thesis, 2000. Thèse de doctorat dirigée par Fouillat, Pascal et Sarger, Laurent Sciences physiques et de l'ingenieur. Instrumentation et mesures Bordeaux 1 2000. (Cité page 24.)

[112] N. Pramstaller, F. K. Gurkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner. Towards an aes crypto-chip resistant to differential power analysis. In *Proceedings of the 30th European Solid-State Circuits Conference*, pages 307–310, Sept 2004. (Cité page 31.)

[113] ARM Cortex-M3 Processor. Technical reference manual. ARM 100165_0201_00_en, r2p1. Copyright © 2005-2008, 2010,2015 ARM. All rights reserved. (Cité page 184.)

[114] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema) : Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards : Smart Card Programming and Security*, E-SMART '01, pages 200–210, London, UK, UK, 2001. Springer-Verlag. (Cité pages 42 et 257.)

[115] X. Ren, V. G. Tavares, and R. D. S. Blanton. Detection of illegitimate access to jtag via statistical learning in chip. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 109–114, March 2015. (Cité page 210.)

[116] S Rimdusit and H Ishida. Development of new class of electronic packaging materials based on ternary systems of benzoxazine, epoxy, and phenolic resins. *Polymer*, 41(22) :7941 – 7949, 2000. (Cité page 235.)

[117] P. Rohatgi. Electromagnetic attacks and countermeasures. In *Cryptographic Engineering*, pages 407–430. Springer, 2009. (Cité page 42.)

[118] Dan Rosenberg. kptr_restrict for hiding kernel pointers from unprivileged users. [Online] Available : https://lwn.net/Articles/420403/, December 2010. (Cité page 207.)

[119] K Rosenfeld and R Karri. Attacks and defenses for jtag. *Design Test, IEEE*, PP(99) :1–1, 2013. (Cité pages 26, 32, 175, 181, 194 et 210.)

[120] D. Samyde, S. Skorobogatov, R. Anderson, and J. J. Quisquater. On a new way to read data from memory. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 65–69, Dec 2002. (Cité page 22.)

[121] Mishra Sanjeeb, Kumar Singh Neeraj, and Rousseau Vijayakrishnan. *System on Chip Interfaces for Low Power Design*. Elsevier, 1st edition, december 2015. (Cité page 247.)

[122] Werner Schindler, Kerstin Lemke, and Christof Paar. *A Stochastic Model for Differential Side Channel Cryptanalysis*, pages 30–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. (Cité page 256.)

[123] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of aes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 41–57, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cité page 23.)

[124] J. M. Schmidt and C. Herbst. A practical fault attack on square and multiply. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 53–58, Aug 2008. (Cité page 13.)

[125] N. Selmane, S. Guilley, and J. Danger. Practical setup time violation attacks on aes. In *2008 Seventh European Dependable Computing Conference*, pages 91–96, May 2008. (Cité page 19.)

[126] Freescale Semiconductor. Configuring secure jtag for the i.mx 6 series family of applications processors. Document Number : AN4686 - Rev. 1, March 2015. © 2015 Freescale Semiconductor, Inc. All rights reserved. (Cité pages 32, 182 et 210.)

[127] A. Shrivastav, G. S. Tomar, and A. K. Singh. Performance comparison of amba bus-based system-on-chip communication protocol. In *2011 International Conference on Communication Systems and Network Technologies*, pages 449–454, June 2011. (Cité page 250.)

[128] M. Da Silva, M. l. Flottes, G. Di Natale, B. Rouzeyre, P. Prinetto, and M. Restifo. Scan chain encryption for the test, diagnosis and debug of secure circuits. In *2017 22nd IEEE European Test Symposium (ETS)*, pages 1–6, May 2017. (Cité page 180.)

[129] Sergei Skorobogatov. *Physical Attacks and Tamper Resistance*, pages 143–173. Springer New York, New York, NY, 2012. (Cité page 32.)

[130] Sergei P. Skorobogatov and Ross J. Anderson. *Optical Fault Induction Attacks*, pages 2–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. (Cité page 24.)

[131] Kenneth C. Smith and Adel S. Sedra. *Micorelectronic Circuits*, chapter "MOS Fiel-Effect Tranistors (MOSFETs)". Oxford University Press, 7th edition, november 2014. (Cité pages 249 et 250.)

[132] M. Soucarros, C. Canovas-Dumas, J. Clédière, P. Elbaz-Vincent, and D. Réal. Influence of the temperature on true random number generators. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 24–27, June 2011. (Cité page 25.)

[133] Starbug and Nohl Karsten. Hardware reverse engineering. 25th. Chaos Communication Congress, December 2008. https://events.ccc.de/congress/2008/Fahrplan/attachments/1218_081227.25C3.HardwareReversing.pdf. (Cité page 265.)

[134] Fuber Steve. *ARM System-on-chip Architecture*. Addison Wesley, 2nd edition, 2000. (Cité pages 4 et 233.)

[135] STMicroelectronics. Proprietary code read out protection on stm32l1 microcontrollers. [Online] Available : https://www.st.com/content/ccc/resource/technical/document/application_note/b4/14/62/81/18/57/48/05/DM00075930.pdf/files/DM00075930.pdf/jcr:content/translations/en.DM00075930.pdf, 2013. (Cité page 183.)

[136] J. Takahashi, T. Fukunaga, and K. Yamakoshi. Dfa mechanism on the aes key schedule. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pages 62–74, Sept 2007. (Cité page 72.)

[137] ARM Security Technology. Building a secure system using trustzone technology. PRD29-GENC-009492C. Copyright © 2005-2009 ARM Limited. All rights reserved. (Cité pages 32, 33, 134 et 252.)

[138] E. Ashfield *et al.* Serial wire debug and the coresight debug and trace architecture. [Online] Available : http://www.arm.com/miscPDFs/15531.pdf. [Accessed] Aug. 28, 2009. (Cité pages 178 et 275.)

[139] Olivier THOMAS. Advanced ic reverse engineering techniques : In depth analysis of a modern smart card. Blackhat USA, August 2015. https://www.blackhat.com/us-15/briefings.html. (Cité pages 8 et 11.)

[140] N. Timmers, A. Spruyt, and M. Witteman. Controlling pc on arm using fault injection. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 25–35, Aug 2016. (Cité pages 13, 16, 119, 122 et 127.)

[141] K. Tobich, P. Maurine, P. Y. Liardet, M. Lisart, and T. Ordas. Voltage spikes on the substrate to obtain timing faults. In *2013 Euromicro Conference on Digital System Design*, pages 483–486, Sept 2013. (Cité page 20.)

[142] Randy Torrance and Dick James. *The State-of-the-Art in IC Reverse Engineering*, pages 363–381. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. (Cité pages 8, 18 et 266.)

[143] J. C. Tsang, J. A. Kash, and D. P. Vallett. Picosecond imaging circuit analysis. *IBM Journal of Research and Development*, 44(4) :583–603, July 2000. (Cité page 23.)

[144] I. Verbauwhede, D. Karaklajic, and J. M. Schmidt. The fault attack jungle - a classification model to guide you. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 3–8, Sept 2011. (Cité page 11.)

[145] Marc Witteman. Advances in smartcard security. *Information Security Bulletin*, 7 :11–22, July 2002. (Cité page 8.)

[146] W. Wolf, A. A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10) :1701–1713, Oct 2008. (Cité page 4.)

[147] Bo Yang, Kaijie Wu, and Ramesh Karri. Scan based side channel attack on dedicated hardware implementations of data encryption standard. In *2004 International Conference on Test*, pages 339–344, Oct 2004. (Cité pages 26 et 175.)

[148] Hiroto Yasuura, Tohru Ishihara, and Masanori Muroyama. *Energy Management Techniques for SOC Design*, pages 177–223. Springer Netherlands, Dordrecht, 2006. (Cité page 247.)

[149] Weilie Zhou, Robert Apkarian, Zhong Lin Wang, and David Joy. *Fundamentals of Scanning Electron Microscopy (SEM)*, pages 1–40. Springer New York, New York, NY, 2007. (Cité page 267.)

[150] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, M. Nicewicz, C. A. Russell, W. Y. Wang, L. B. Freeman, P. Hosier, L. E. LaFave, J. L. Walsh, J. M. Orro, G. J. Unger, J. M. Ross, T. J. O'Gorman, B. Messina, T. D. Sullivan, A. J. Sykes, H. Yourke, T. A. Enger, V. Tolat, T. S. Scott, A. H. Taber, R. J. Sussman, W. A. Klein, and C. W. Wahaus. Ibm experiments in soft fails in computer electronics (1978-1994). *IBM Journal of Research and Development*, 40(1) :3–18, Jan 1996. (Cité page 24.)

[151] Loic Zussa, Jean-Max Dutertre, Jessy Clédiere, Bruno Robisson, Assia Tria, et al. Investigation of timing constraints violation as a fault injection means. In *27th Conference on Design of Circuits and Integrated Systems (DCIS), Avignon, France*, 2012. (Cité page 23.)

Annexe A

Partitionnement fonctionnel des SoC

Dans cette annexe, nous nous efforçons de rappeler brièvement les caractéristiques des SoC tels que ceux présents sur le marché actuel. Loin d'être aussi exhaustifs que des ouvrages tels que [53, 81, 134], ou même de la documentation constructeur, cette description vise à mettre en avant la diversité technologique présente sur ces systèmes aux architectures hétérogènes et modulaires.

Sommaire de l'annexe

Pour décrire les architectures de ces systèmes polyvalents composées de différents modules, nous pouvons considérer divers sous-ensembles consacrés à des tâches spécifiques [53]. Parmi ceux-ci, les plus représentatifs sont :

1. Les boîtiers.
2. Les processeurs ou *Central Processing Unit* (CPU).
3. Les blocs mémoires.
4. La gestion de la puissance et des horloges.
5. Les bus qui relient les différents blocs.
6. Les divers périphériques.
7. Le débogage matériel.
8. La sécurité.

Tous ces modules ont pour objectif d'augmenter les fonctionnalités proposées par un unique système. Il est à noter que les points 7. et 8. ne sont pas développés ici étant donné que le chapitre 5 et la section 1.3 y sont respectivement consacrés.

A.1 Les boîtiers

Les systèmes sur puce sont destinés à être intégrés dans des appareils portatifs, c'est pourquoi leur conditionnement au sein des boîtiers est primordial pour satisfaire les contraintes d'encombrement, de dissipation thermique et de protection mécanique. Plusieurs technologies ont été développées pour les boîtiers des SoC de *smartphones* et nous présentons ici, celles qui sont les plus répandues sur le marché des systèmes embarqués actuels.

A.1.1 Matériaux

Les matériaux utilisés pour les boîtiers sont généralement des résines, du plastique ou du métal [116, 82]. La résine époxy (principalement la FR-4) constitue le PCB qui structure la matrice de billes sur laquelle sont intégrés les circuits ASIC du SoC. En fonction des modèles, le circuit peut être en face avant ou face arrière et peut avoir une protection en plastique ou en métal. En fonction des besoins de refroidissement, si la protection est en métal, on parle de dissipateur thermique intégré et, en l'absence de protection et avec l'ajout d'une structure métallique caloporteuse destinée au refroidissement, on emploie alors les termes de radiateur ou de dissipateur thermique.

A.1.2 Câblage de la puce

Plusieurs technologies sont utilisées pour connecter la puce avec l'extérieur à travers le boîtier de SoC[1]. Parmi celles-ci, on trouve deux techniques : le câblage par fil (*wire*

1. https://perso.esiee.fr/~vasseurc/assemblage.html

bonding en anglais) et la technologie dite « retournée » (*flip chip* en anglais). Le câblage par fil consiste à connecter des fils entre le circuit intégré et les broches de son boîtier (voir Fig. A.1). A l'inverse, les puces utilisant la technologie « retournée », auront la face active directement soudée sur des plots pour faire la connexion avec l'extérieur (voir Fig. A.2). Cette seconde technique est largement utilisée dans les SoC car elle permet une meilleure répartition des plots ainsi qu'un plus grand nombre de connexions sur la surface de la puce. Notons, qu'indépendamment de la technologie utilisée, la face active de la puce peut être orientée selon les deux directions : vers le circuit imprimé ou vers l'extérieur. Le choix dépend des besoins de la conception.

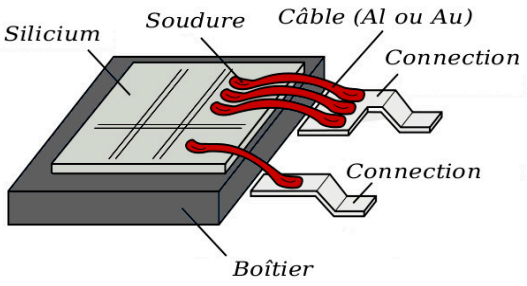


FIGURE A.1 – Technologie *wire bonding*[2]

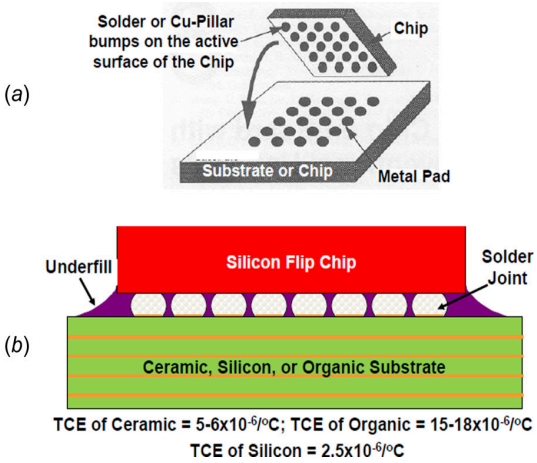


FIGURE A.2 – Technologie *flip chip*[3]

A.1.3 Les matrices de billes

Le *Joint Electron Device Engineering Council* (JEDEC) décrit ce type de boîtiers par une face recouverte d'un quadrillage de billes de soudure qui fera office de connexions entre un circuit intégré (SoC) et un circuit imprimé (appareil mobile)[7]. Le pas entre chaque bille est de l'ordre du dixième de millimètre (voir Fig. A.3). Ces boîtiers, aussi connus sous leurs dénomination anglaise : *Ball Grid Array* (BGA), apportent une partie de la solution au problème d'encombrement en proposant des circuits intégrés disposant de plusieurs centaines de connexions dans un minimum d'espace. De plus, leurs billes diminuent la résistance des connexions, ce qui a pour résultat d'abaisser la température d'échauffement durant le fonctionnement. Elles réduisent également l'inductance et la capacitance qui déforment les signaux échangés entre le SoC et le circuit imprimé. Les systèmes équipés avec de tels boîtiers ont des performances électriques supérieures à ceux équipés avec des boîtiers à pattes.

2. Source : https://en.wikipedia.org/wiki/Wire_bonding
3. Source : [74]

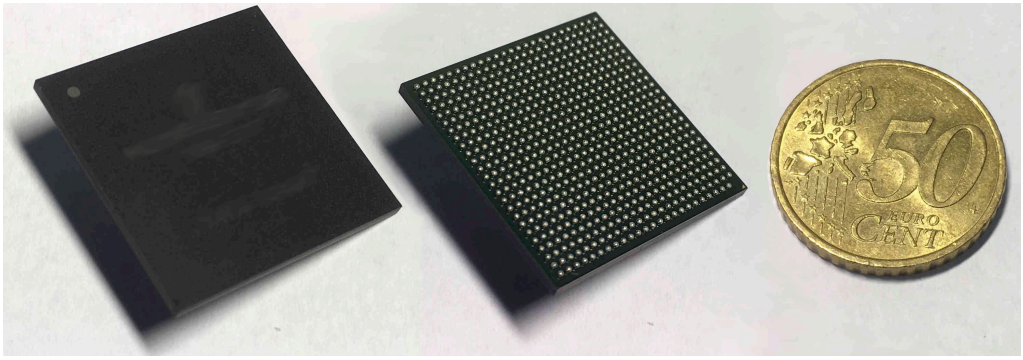


FIGURE A.3 – Boîtier BGA de 25 x 25 billes avec un pas de 0.8 mm d'un SoC

A.1.4 Package-on-Package

L'idée de combiner l'ensemble des éléments d'un système (parties logiques et analogiques, mémoires et autres composants) dans un même boîtier a émergé avec le développement des systèmes complexes embarqués. Dans les étapes d'augmentation de la densité d'intégration, le SoC vient après le microcontrôleur. Pour augmenter encore le niveau d'intégration, l'utilisation de boîtiers du type *Package-on-Package* (PoP) s'est démocratisée en suivant des standards comme ceux proposés par le JEDEC [5]. La méthode de conditionnement PoP consiste à empiler les différents sous-systèmes d'une certaine manière pour gagner un maximum de place. La figure A.4 décrit un PoP sur lequel deux couches sont empilées. Il est très courant que les parties logiques (ex. processeurs), qui nécessitent beaucoup de connexions, soient placées sur la couche inférieure, *i.e.* celle qui est la plus proche des billes et que les systèmes nécessitant moins de connexions (ex. mémoires) soient empilés sur les couches supérieures. La technique *Package-on-Package* essaye de combiner les bénéfices des boîtiers traditionnels avec ceux de l'empilage de composants en évitant les effets indésirables. Outre le gain d'espace, cette technique apporte également des avantages pour les performances électriques des composants. Comme pour les matrices de billes décrites ci-dessus, les courtes distances entre chaque élément diminuent la résistance, l'inductance et la capacitance des liaisons. Ceci permet une propagation plus rapide des signaux et une réduction du bruit. Cette méthode de conception représente également un avantage économique du point de vue des industriels. Le fait que les éléments soient découplés permet, d'une part, aux fabricants de les choisir chez différents fournisseurs et, d'autre part, de les tester séparément avant l'assemblage final et ainsi d'écarter uniquement ceux qui présentent des dysfonctionnements.

A.2 Les processeurs

Aujourd'hui, la majorité des processeurs embarqués sont ceux conçus à partir d'IP proposés par la socitété ARM®. Il s'agit de processeurs ayant des architectures *Reduced Instruction-Set Computer* (RISC) en opposition avec les architectures *Complex Instruction-Set Computer* (CISC). Les principes généraux des processeurs RISC sont [67] :

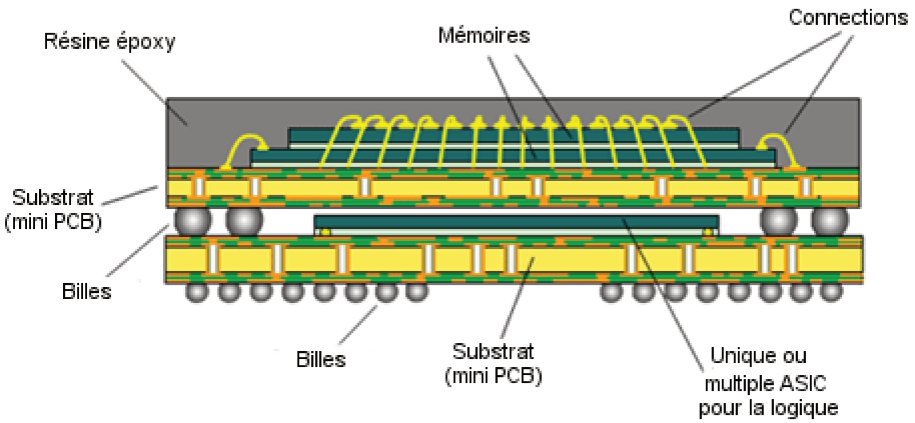


FIGURE A.4 – *Package-on-Package* typique avec de la mémoire superposée sur la partie logique

- Un ensemble restreint d'instructions élémentaires. Ceci facilite le décodage et simplifie la gestion de la mémoire.
- Chaque instruction est exécutée en un cycle d'horloge. Celles nécessitant plusieurs cycles (ex. virgule flottante) sont exécutées par un coprocesseur afin de ne pas ralentir le processeur principal.
- Les instructions ont un format et une taille fixe. Ceci permet un décodage simplifié.
- Seules les instructions `LDR` (*load register*, charger un registre en français) et `STR` (*store register*, stocker un registre) ont accès à la mémoire. Elles utilisent l'indirection avec des registres pour manipuler des données en mémoire. Cette contrainte simplifie l'architecture du processeur.

Le monopole d'ARM® dans les microprocesseurs embarqués peut s'expliquer à partir de l'année 1998. C'est à cette date que ARM® sort son premier processeur destiné aux SoC. Il s'agit de l'ARM7TDMI qui est une extension de l'ARM7. Ce processeur inclue de nouvelles spécificités matérielles. En premier lieu, pour le débogage, les initiales « D » et « I » correspondent respectivement à *Debug* et *In-Circuit Emulator* qui sont deux techniques de débogage que nous développerons dans le Chapitre 5. Ce processeur inclue également des spécificités qui augmentent les performances des systèmes embarqués : l'initial « T » fait référence à un nouveau jeu d'instructions compressées dénommées THUMB. Ces instructions permettent aux développeurs de logiciels de loger plus de code dans un espace mémoire donné ou de réduire la taille totale de la mémoire requise. Enfin, ce processeur intègre un large multiplieur matériel dénoté par la lettre « M » pour appliquer plus rapidement les algorithmes liés aux traitements des signaux analogiques par exemple. Avec des circuits intégrés à faible surface, basse consommation et dotés de riches ensembles d'instructions, ces processeurs ont rapidement suscités un vif intérêt chez les industriels de téléphones portables de l'époque. L'ARM7TDMI a rapidement été adopté comme processeur pour les

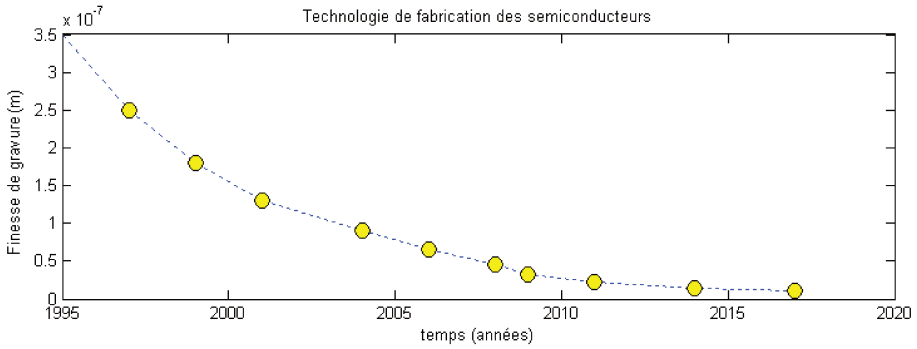


FIGURE A.5 – Evolution de la finesse de gravure des processeurs Intel depuis 20 ans [4]

mobiles. Dès lors, les processeurs ARM® n'ont pas cessé d'évoluer en ajoutant de nouvelles spécificités et fonctionnalités pour répondre à la demande d'un marché toujours plus exigeant.

Parmi les améliorations, on trouve l'apparition d'un élément visant à accélérer l'exécution des instructions : le *pipeline*. Il s'agit d'un ensemble de registres indépendants montés en séries pour commencer l'exécution d'une instruction sans attendre que la précédente soit terminée. Chaque registre est appelé « étage ». Le *pipeline* classique RISC possède 5 étages qui permettent d'exécuter 5 instructions en 9 cycles d'horloge au lieu de 25 cycles. D'autre part, les améliorations sont aussi liées aux technologies de réalisation des processeurs. Sans entrer dans les détails, nous pouvons constater que le nombre de transistors implantés sur le silicium des puces à tendance à suivre la conjecture de Moore. Intel, le leader mondial des microprocesseurs, produit des processeurs avec des portes logiques de plus en plus petites. La figure A.5 illustre cette évolution.

Aujourd'hui, les processeur RISC n'ont presque plus rien à voir avec avec leurs acronymes. Les principes originaux, comme par exemple celui qui limitait la quantité totale d'instructions, ont été délaissés pour ajouter une multitude d'instructions répondant à la demande du marché. Les processeurs ARM® actuels se déclinent en architectures 32 bits (ARMv7) ou 64 bits (ARMv8) et travaillent souvent en parallèle avec plusieurs autres processeurs cadencés à plusieurs gigahertz. La figure A.6 illustre les ajouts de fonctionnalités que l'on a pu voir apparaitre depuis le premier processeur ARM® destiné aux SoC. Le tableau A.2, fourni une liste de SoC ayant été largement utilisés en 2016 dans l'industrie des mobiles. Le marché d'aujourd'hui embarque des systèmes sur puce ayant un niveau de complexité de plus en plus élevé.

A.2.1 Registres et modes de fonctionnement.

Les cœurs ARM antérieur à la version ARMv6 ont sept modes de fonctionnement. A partir d'ARMv6 qui implémente le TrustZone®, un mode a été ajouté :

- **Le mode user :** Un mode non privilégié dans lequel la plupart des programmes s'exé-

4. Source :http://www.intel.com

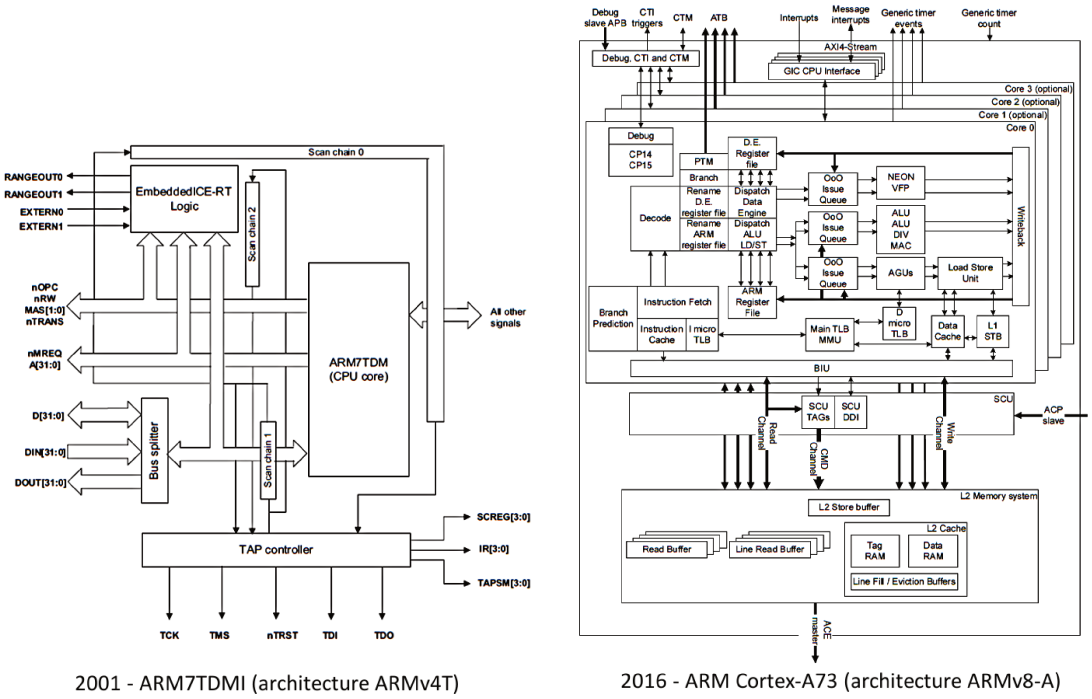


FIGURE A.6 – Comparaison des schémas ARM®7TDMI et ARM®Cortex-A73

Marque	Modèle	cœurs	Q^{té}	a	b	c
Qualcom	Snapdragon 821	Kryo	4	2.4	ARMv8-A	14
Samsung	Exynos 8890	M1	4	2.3	ARMv8-A	14
MediaTek	Helio X25	Cortex-A53	4	1.6	ARMv8-A	
		Cortex-A72	2	2.5	ARMv8-A	
		Cortex-A53	4	2.0	ARMv8-A	20
		Cortex-A53	4	1.55	ARMv8-A	
HiSilicon	Kirin 955	Cortex-A72	4	2.5	ARMv8-A	14
NXP	i.MX8	Cortex-A53	4	2.0	ARMv8-A	
		Cortex-A53	4	1.2	ARMv8-A	
		Cortex-A72	2	1.8	ARMv8-A	28
		Cortex-M4	2	0.266	ARMv7-M	

a. Fréquence maximum de calcul en gigahertz

b. Architecture de processeur ARM®

c. Technologie de gravure en nanomètre

Tableau A.1 – SoC embarquant des processeurs ARM® largement vendus en 2016 dans l'industrie des *smartphones*

System et User	FIQ	Supervisor	Abort	IRQ	Undefined	Secure Monitor
r0	r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11	r11
r12(ip)	r12_fiq(ip)	r12(ip)	r12(ip)	r12(ip)	r12(ip)	r12(ip)
r13(sp)	r13_fiq(sp)	r13_svc(sp)	r13_abt(sp)	r13_irq(sp)	r13_und(sp)	r13_sm(sp)
r14(lr)	r14_fiq(lr)	r14_svc(lr)	r14_abt(lr)	r14_irq(lr)	r14_und(lr)	r14_sm(lr)
r15(pc)	r15(pc)	r15(pc)	r15(pc)	r15(pc)	r15(pc)	r15(pc)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und	SPSR_sm

FIGURE A.7 – Banque de registres des processeurs ARMv6. Les registres qui conservent leurs valeurs pour un état donné sont notés en bleu.

cutent.

- **Le mode system :** Le mode dans lequel est exécuté le système d'exploitation.
- **Le mode FIQ :** Un mode privilégié dans lequel le processeur entre lorsqu'il accepte une interruption FIQ.
- **Le mode Supervisor :** Un mode protégé pour le système d'exploitation. Etat obtenu avec un `svc` (*supervisor call*) ou après un `reset`.
- **Le mode Abort :** Un mode privilégié dans lequel le processeur entre lorsque le signal `abort` est détecté.
- **Le mode IRQ :** Un mode privilégié dans lequel le processeur entre lorsqu'il accepte une interruption IRQ.
- **le mode undefined :** Un mode privilégié dans lequel le processeur entre lorsqu'une instruction inconnue est exécutée.
- **le mode secure monitor :** Présent à partir des architectures ARMv6 avec TrustZone®. C'est le mode de fonctionnement privilégié très sensible d'un point de vue sécurité car il permet d'effectuer une changement de contexte entre le *Normal World* et le *Secure World*.

Chaque état de fonctionnement dispose de registres 32-bits pouvant être utilisés pour des fonctions particulières (voir Fig. A.7). Les modes *user* et *system* possèdent 16 registres.

Les registres `r0` à `r11` sont des registres généraux qui peuvent être utilisés pour n'importe quelle opération. Le registre `r12 (ip)` est l'*intraprocedure register*, il sert à stocker temporairement des données lorsque l'on passe d'une fonction à une autre. Le registre `r13 (sp)` est le *stack pointer* qui indique l'adresse du haut de la pile. Le registre `r14 (lr)` est le *link register* qui contient l'adresse de retour lorsqu'une fonction est appelée. Le `r15 (pc)` est le *program counter* qui indique l'adresse de la prochaine instruction à exécuter. Enfin le registre `CPSR`, *current program status register* est un registre spécial mis à jour par le biais de différentes instructions. Il sert par exemple, pour les instructions conditionnelles, et enregistre le mode d'exécution actuel. Les six autres modes ont en plus le registre `SPSR` (*Saved Program Status Register*) qui est destiné à stocker la valeur du `CPSR`.

A.3 Les mémoires

Avec le processeur, les mémoires constituent la partie essentielle au fonctionnement d'un système informatique. La figure A.9 décrit l'implantation typique des différentes mémoires dans un SoC. Elles peuvent être classées suivant le diagramme de la figure A.8. Chacune d'elles possède des spécificités adaptées aux différentes tâches qu'un système sur puce devra traiter. Les mémoires non volatiles ont la particularité de conserver les données lorsqu'elles ne sont plus alimentées. Certaines d'entre elles ont des temps d'accès relativement longs avec un débit de données limité par rapport à la vitesse de calcul du processeur. A l'opposé, les mémoires volatiles proposent des temps d'accès bien plus courts avec des débits bien plus élevés mais perdent leurs données lorsqu'elles ne sont plus alimentées. En fonction de la technologie, il peut y avoir une différence d'un facteur 10^6 entre le temps de réponse des mémoires volatiles les plus rapides et celui des mémoires non volatiles les plus lentes (ex. registres CPU et disques durs, voir Fig.A.10). Un système a certes besoin de mémoires persistantes pour conserver des données essentielles à son fonctionnement mais il doit également être capable d'effectuer des opérations en un temps restreint pour avoir des performances satisfaisantes. Il faut noter que plus une technologie de mémoire est rapide d'accès, plus elle est chère à fabriquer et par conséquent plus elle sera limitée en taille de stockage. C'est pourquoi afin d'avoir des systèmes performants à des coûts acceptables, les concepteurs doivent trouver des architectures qui utiliseront de manière optimale les différents types de mémoires avec le processeur [84, 93].

A.3.1 Les Mémoires non volatiles

A.3.1.1 La Mémoire Boot ROM

La mémoire Boot ROM est une mémoire à lecture seule ou *Read-Only Memory* (ROM). Elle conserve de manière permanente les données qui y sont inscrites lors de sa fabrication. Elle est utilisée dans les SoC pour stocker les quelques kilooctets de code qui seront exécutés au démarrage, *i.e.* juste après la mise sous tension. Ce code connu sous le nom de *Boot ROM code* ou code d'amorçage en français, contient notamment la table des vecteurs d'interruptions. Il est nécessaire pour initialiser le système et charger les autres portions de codes qui poursuivront le démarrage complet du système. Durant son exécution, le paramétrage de certaines fonctionnalités et des niveaux de sécurité est également effectué

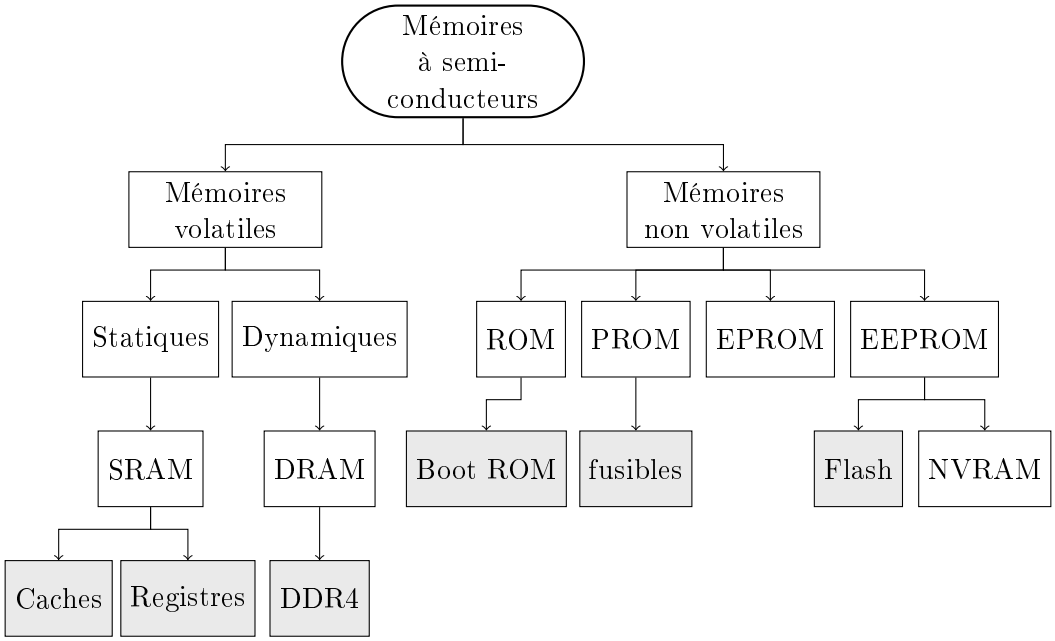


FIGURE A.8 – Les types de mémoires à semi-conducteurs. Celles présentes dans le SoC utilisées dans cette étude sont grisées

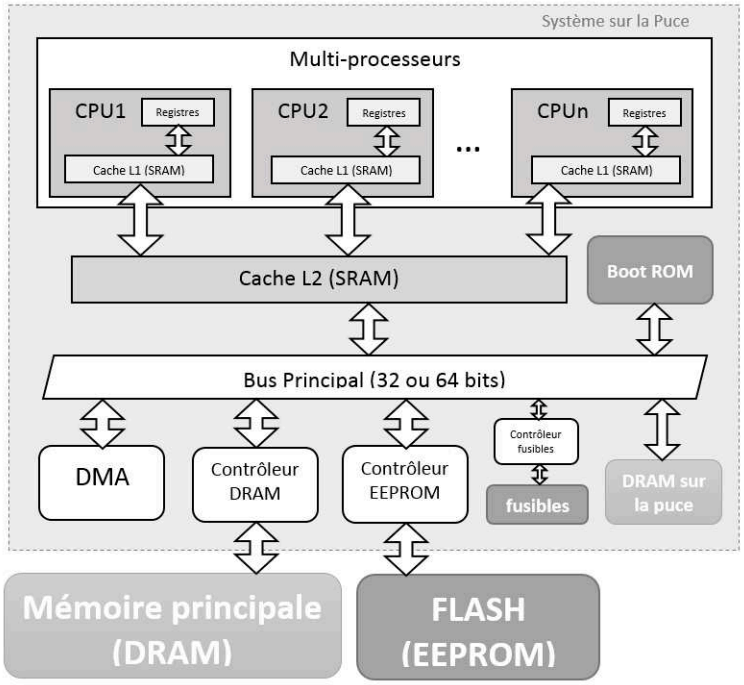


FIGURE A.9 – Les mémoires des systèmes sur puce

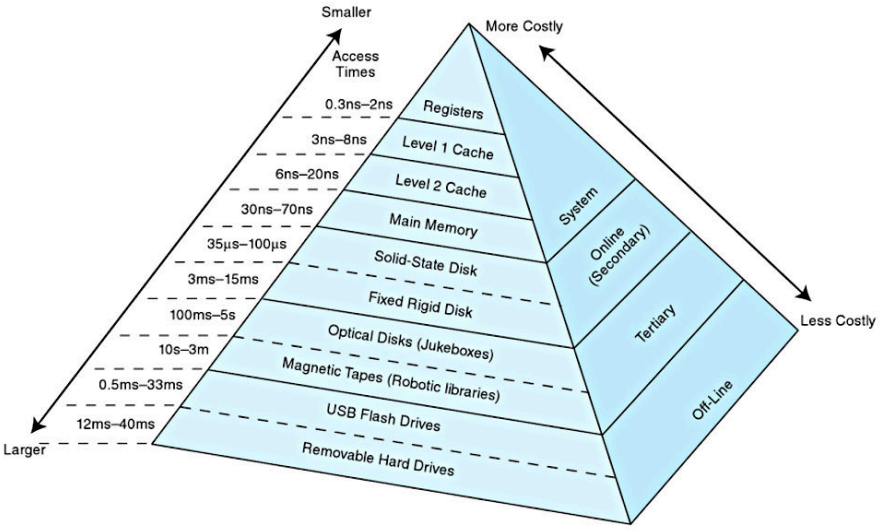


FIGURE A.10 – Hiérarchie des mémoires (Source : [84])

en s'appuyant sur des valeurs que les développeurs peuvent définir. Ces valeurs sont souvent enregistrées dans un autre type de mémoire interne, les fusibles.

A.3.1.2 Les fusibles

Les fusibles sont des mémoires de la catégorie des mémoires programmables à lecture seule, *Programmable Read-Only Memory* (PROM). Ils sont conçus pour être programmés une seule fois et ne contiennent que quelques kilooctets de données. Les configurer altère physiquement leur état, ce qui leur fait conserver une valeur de manière définitive. Ils font partie des solutions technologiques dites *One-Time Programmable* (OTP), programmables qu'une fois en français. Avant que les SoC ne soient commercialisés, les fabricants « grillent » les fusibles OTP afin de mémoriser définitivement des valeurs qui seront utilisées lors des prochains démarrages de l'appareil. Parmi ces valeurs, figurent notamment : un numéro de série, une clef maître de chiffrement et un niveau de privilège pour accéder au débogage. On trouve également l'information de la mémoire sur laquelle le *Boot ROM code* ira chercher le système d'exploitation. Cette donnée désigne une mémoire EEPROM qui souvent est une mémoire flash.

A.3.1.3 La mémoire flash

La mémoire flash est externe au SoC. Elle est généralement soudée sur le circuit imprimé. C'est une mémoire du type *Electrically Erasable Programmable Read-Only Memory* (EEPROM), en français mémoire électriquement programmable ou effaçable à lecture seule. Il existe deux grands types de mémoire flash : les flashs NOR et les flashs NAND, selon le type de porte logique utilisée pour les cellules de stockage. Schématiquement, l'architecture des flashs NOR permet d'accéder à n'importe laquelle de ses cellules mémoires tandis que celle des NAND impose une lecture séquentielle de ses données. Cependant, la rapidité de

lecture et d'écriture de ces dernières font que cette gestion séquentielle n'est pas tellement moins performante.

Le faible coût et la densité supérieure de stockage de la NAND font qu'elle est largement utilisée contenir le système d'exploitation du SoC. La technologie de ces mémoires leur permet de stocker une grande quantité de données (plusieurs centaines de gigaoctets) et d'être facilement reprogrammables. Pendant le démarrage du système, pour des raisons de performances, le *Boot ROM code* copie le code du système d'exploitation depuis la flash vers une mémoire volatile du système sur laquelle le processeur travaillera. Cette dernière est la mémoire principale du système.

A.3.2 Les Mémoires volatiles

Les mémoires volatiles actuelles offrent d'excellentes performances en matière de temps de réponse et de débit de transfert. De plus, la répétition des tâches n'altère pas ces dernières durant la vie du système. C'est pourquoi ce sont elles qui interagissent directement avec le processeur, en particulier la mémoire principale avec laquelle il travaille en chargeant et stockant des données en permanence. Cette mémoire doit avoir une taille et des performances suffisantes pour que les programmes s'exécutent de manière fluide. Bien qu'il soit possible de produire des mémoires quasiment aussi rapides que les processeurs, cela se fait avec de coûts de fabrication qui demeurent très élevés. Et, en outre, il peut persister des temps de latence car la communication entre les différents éléments est assurée par l'intermédiaire de bus et de contrôleurs. Par conséquent, les concepteurs de processeurs préfèrent des architectures de mémoires moins onéreuses mais plus astucieuses. Pour cela ils utilisent des mémoires principales plus lentes et compensent ce ralentissement par l'ajout de plusieurs niveaux de mémoires plus rapides jusqu'au processeur. Ces mémoires sont agencées selon la hiérarchie présentée figure A.10, qui exploite leurs caractéristiques de manière optimale.

A.3.2.1 La mémoire principale

La mémoire principale, dite aussi mémoire vive, est une mémoire volatile dynamique. Elle est également connue sous sa dénomination anglaise *Dynamique Random Access Memory* (DRAM) et est parfois simplement appelée RAM. Ces mémoires présentent les avantages d'être compactes, peu coûteuses, avec des capacités allant jusqu'à une dizaine de gigaoctets. Elles sont constituées de plusieurs millions de pico-condensateurs dont chaque charge représente un bit de mémoire. Le terme dynamique vient du fait que la charge d'un condensateur diminue dans le temps et qu'il est nécessaire de la rafraîchir régulièrement pour pouvoir conserver la donnée. Dans cet optique, on couple des transistors aux condensateurs afin de régulièrement actualiser la valeur. Comme illustré dans la figure A.11, les transistors sont rangés pour former une matrice dont il faut spécifier la ligne et la colonne pour accéder à 1 bit mémoire. Cette mémoire peut, soit être empilée sur le SoC comme nous l'avons vu dans le paragraphe A.1 et dans ce cas nous parlons d'une architecture PoP, soit être complètement extérieure au boîtier. Lorsqu'un programme doit s'exécuter, celui-ci est copié depuis la flash vers la RAM qui possède de meilleures performances d'accès.

Le type de mémoire DRAM est aussi souvent utilisé en tant que RAM interne. Il s'agit généralement d'un bloc de mémoire RAM réduit à une centaine de kilooctets qui sera

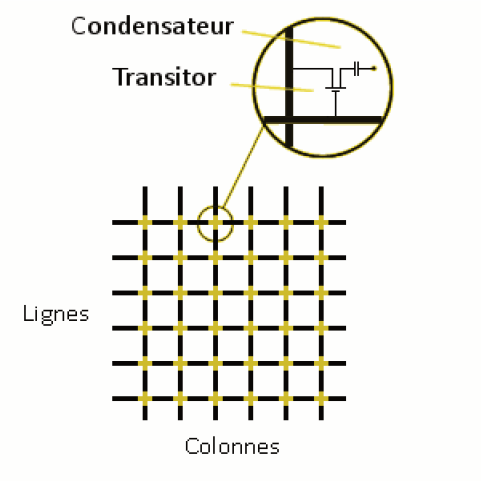


FIGURE A.11 – Structure matricielle des mémoires DRAM

employé pour stocker des informations sensibles dont le processeur se sert régulièrement. Cette mémoire peut contenir des données utiles au CPU, comme la *stack* par exemple, ou des données qui ne doivent pas être corrompues, telles que les tables de l'unité de gestion de la mémoire (MMU).

A.3.2.2 Les mémoires cache

Les mémoires cache ou simplement caches sont des mémoires du type *Statique Random Access Memory* (SRAM), mémoires volatiles statiques en français. Les données sont stockées dans un ensemble de bascules qui, contrairement aux mémoires dynamiques n'ont pas besoin d'être rafraîchies pour conserver leur état. Cette technologie offre une vitesse d'exécution aussi rapide que celle des processeurs et consomme moins d'énergie que la RAM. En revanche, elle est largement plus chère, ce qui justifie la limitation de la taille de ces mémoires (plusieurs kilooctets). Comme représenté sur les figures A.9 et A.10, les SoC actuels possèdent généralement deux niveaux de caches notés L1 et L2 qui font office de tampon entre le CPU et la mémoire principale. Pour résumer leur fonctionnement, les caches contiennent la copie du code d'un programme présent dans la RAM, que le processeur souhaite exécuter. Pour optimiser cette exécution, il faut que le code soit chargé dans les caches avant que le processeur en ait besoin. Pour cela, les algorithmes utilisés travaillent avec les principes de localité spatiale et temporelle. La localité spatiale se base sur le fait que, lorsque l'adresse d'une instruction est exécutée par le CPU, il existe une forte probabilité pour que les suivantes se situent à des adresses proches. Il en est de même pour la localité temporelle, qui s'appuie sur la forte probabilité qu'une zone mémoire dans laquelle se trouve la donnée en train d'être utilisée par le processeur, soit la même pour les prochaines instructions. Ainsi, lorsque le processeur a besoin d'une valeur, le cache vérifie s'il possède cette valeur. Si c'est le cas, il la transmet (réussite), sinon il la demande à la mémoire principale (échec). La cache L2, d'une centaine de kilooctet est souvent commune aux différents processeurs et possède des copies complètes de portions de codes. La cache

L1 d'une dizaine de kilooctet est intégrée dans le processeur. Cette cache est la plus rapide, elle travaille directement avec les registres du CPU et distingue les instructions des données.

A.3.2.3 Les registres

Différents types de registres sont utilisés dans les SoC. Parmi ceux-ci, on trouve des registres utilisés pour les calculs du processeur et d'autres destinés à stocker des valeurs d'état de différents contrôleurs. Dans les deux cas, il s'agit de petits blocs de mémoires à accès très rapide. Les registres CPU de 32 ou 64 bits, en fonction de l'architecture du SoC, sont directement intégrés dans le processeur et travaillent à la même vitesse que lui. Ils servent pour stocker temporairement des valeurs intermédiaires aux calculs et aux données. Des registres peuvent aussi être employés pour définir la configuration d'un contrôleur et l'état dans lequel il travaille. Par exemple, le contrôleur de génération et de gestion des horloges offre une liste de registres permettant notamment de lire ou de sélectionner la fréquence à laquelle travaille le processeur principal. La plupart de ces valeurs sont initialisées lors du démarrage du système par la séquence de *boot* et pour, des raisons de sécurité, sont protégés par des accès restreints.

A.4 La gestion de la puissance et des horloges

Les systèmes sur puce ont pour objectif d'exécuter un ensemble d'applications qui doivent toujours être actives et dont la faible consommation énergétique prolonge la durée de vie des batteries. Ainsi, leur architecture est conçue pour réduire la consommation de courant tout en permettant des pics de performance lorsque cela est nécessaire. Avec la combinaison de techniques à travers différents modules, les SoC optimisent le compromis entre la puissance nécessaire et les fréquences de travail [148, 121]. De plus, il est nécessaire qu'ils puissent générer des signaux d'horloge adaptés à la diversité des modules qu'ils embarquent. Suivant l'architecture du SoC considéré, les besoins exprimés précédemment peuvent être satisfaits par un ou plusieurs contrôleurs. La figure A.12 présente un exemple de réseau de gestion de la puissance et des horloges caractéristique des systèmes sur puce actuels. Sur celle-ci, on observe 4 modules principaux (surlignés en cyan) qui remplissent les fonctions suivantes :

- *Clock Generation*, générateur d'horloge en français qui à partir d'oscillateurs (OSC), de boucles à phases asservies (PLL) et de diviseurs de fréquences (PFD) génère différents arbres d'horloge `Clock i`. Chaque arbre définit une horloge qui sera utilisée comme référence par les différentes zones du SoC.

- *Clock Controller Module* (CCM), contrôleur d'horloge en français qui, à partir des horloges de référence `Clock i` fournies par le module précédent, génère et contrôle les différentes horloges qui serviront pour l'ensemble des modules du système `Root j`.

- *Power Management Unit* (PMU), unité de gestion de puissance en français qui fait l'interface entre l'alimentation électrique extérieure et le système. Il est composé des

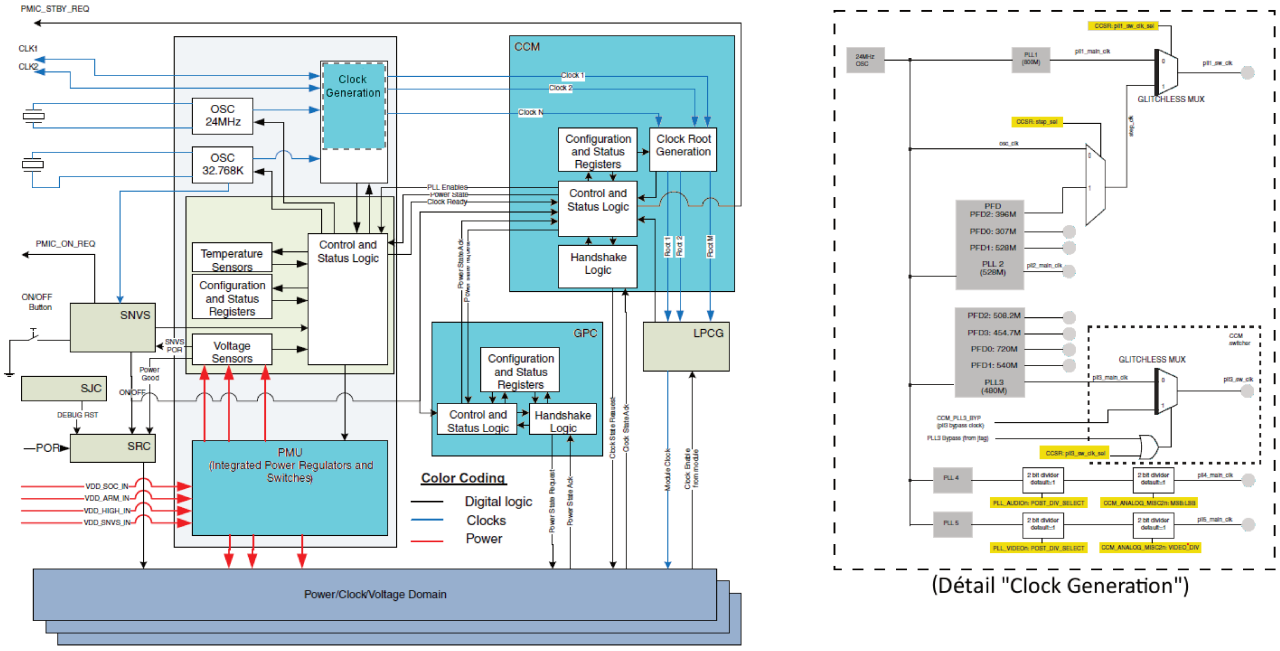


FIGURE A.12 – Gestion de la puissance et des horloges dans SoC (source [94])

sources d'alimentation d'une part, des transformateurs de puissance intégrés et des éléments de contrôle d'autre part. Cette unité intègre également des modules de réduction de la consommation basés sur le *Body Biasing*. Ceci est expliqué dans les paragraphes suivants.

- *General Power Controller* (GPC), contrôleur principal de puissance en français qui gère les modes d'alimentation de tous les modules. Dans cet exemple, c'est lui qui est responsable du *Dynamic Voltage & Frequency Scaling* (DVFS) et du contrôle de l'alimentation des autres modules. Le DVFS est une technique de gestion de puissance que nous expliquons dans la suite.

A.4.1 Le Body Biasing

Dans les transitors MOSFET, le *body effect* est l'influence de la tension source-substrat sur la tension de seuil du transistor. Ceci a pour conséquence d'agir sur le courant dans le drain i_D. Le *body effect* sur la tension de seuil V_{th} d'un transistor NMOS, représenté figure A.13 peut être modélisé par l'équation A.1.

$$V_{th} = V_{th_0} + \gamma\left(\sqrt{2\Phi_f + V_{SB}} - \sqrt{2\Phi_f}\right) \; ; \qquad \gamma = \frac{\sqrt{2\varepsilon_S q N_A}}{C_{OX}} \tag{A.1}$$

Dans celle-ci, V_{th_0} est la tension de seuil lorsque $V_{SB} = 0$ et $2\Phi_f$ le potentiel de la surface (en V). γ est le paramètre de *body effect* (en $V^{1/2}$) où q est la charge élémentaire, N_A est la densité du dopage du substrat de type P, ε_S la permittivité du silicium et C_{OX} la

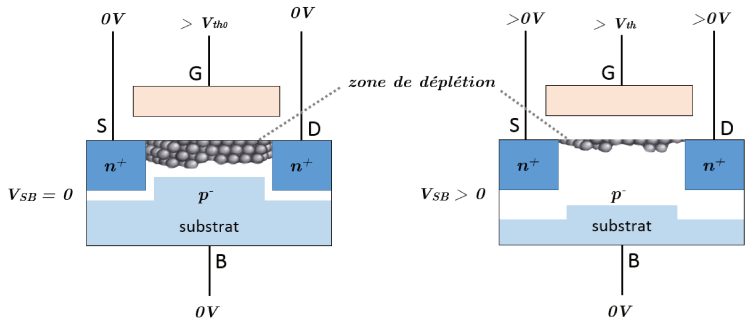


FIGURE A.13 – Body effect sur la tension de seuil d'un transitor NMOS

capacité d'oxyde de grille par unité d'aire. Ceci est valable également pour les transistors PMOS en remplaçant V_{SB} par V_{BS} ou $|V_{SB}|$ et N_A par N_D, la densité du dopage du substrat de type N. Dans ce cas $\gamma < 0$.

Ainsi, selon cette équation, une augmentation de la tension V_{SB} implique une hausse de la tension V_{th} nécessaire pour rendre un transistor passant. Ceci décroît également le courant dans le drain i_D, même si la tension V_{GS} reste constante. Le *body effect* influence les performances de consommation et de vitesse de commutation des transistors [131] (Part I, Ch.5).

Cette propriété est utilisée dans les SoC pour optimiser ces performances au travers du *Reverse Body Biasing* (RBB) et du *Forward Body Biasing* (FBB). Le RBB est un effet qui diminue les courants de fuites i_D et augmente la tension de seuil V_{th} en appliquant une tension négative sur le substrat ($V_{SB} < 0$). A l'inverse, le FBB est un effet qui augmente les courants de fuites i_D et diminue la tension de seuil V_{th} en appliquant une tension positive au substrat ($V_{SB} > 0$).

En utilisant les effets RBB-FBB il est possible de faire varier la tension V_{th} et donc de modifier de manière exponentielle la consommation statique des circuits. Cette consommation dépend de la tension d'alimentation V_{DD} et du courant de fuite. Elle peut être modélisée par [36] :

$$P_{stat} \propto V_{DD} \cdot k_{design} \cdot k_{tech} \cdot 10^{-\frac{V_{th}}{S_T}} \tag{A.2}$$

avec k_{design}, k_{tech} et S_T trois paramètres dépendants de la technologie des circuits. Dans les SoC, on trouve différentes applications de ces effets avec notamment le *Dynamic Body Biasing* Il s'agit d'une technique basée sur les effets RBB-FBB. Elle consiste à ajuster en permanence les potentiels appliqués aux substrats des transistors pour optimiser le compromis performance/puissance. Un autre exemple est l'utilisation du RBB durant les modes *standby* afin de pouvoir couper l'alimentation des transistors inactifs tout en conservant leur état logique [73]. Ces techniques sont souvent combinées avec celle du DVFS [72] pour les raisons expliquées dans le paragraphe suivant.

A.4.2 DVFS

Le *Dynamique Voltage & Frequence Scaling* est une technique largement utilisée pour diminuer la consommation des SoC actuels. Elle adapte la fréquence d'horloge et la tension

d'alimentation en fonction des tâches à effectuer. Ceci a pour objectif de minimiser la consommation dynamique de puissance définie par [131] :

$$P_{dyn} \propto C_{sw} \cdot f \cdot V_{DD}^2 \tag{A.3}$$

avec f la fréquence de fonctionnement et V_{DD} la tension d'alimentation du circuit. C_{sw} représente la quantité moyenne de capacités qui commutent à chaque cycle d'horloge. Cette puissance dynamique est linéaire par rapport à la fréquence mais quadratique vis-à-vis de l'alimentation.

Le compromis entre consommation et performance est déterminé par un contrôleur matériel (le GPC, dans l'exemple de la figure A.12), qui consulte régulièrement les recommandations données par un algorithme implémenté le plus souvent dans le système d'exploitation.

Cependant, la technique du DVFS est plus compliquée qu'il n'y paraît. En effet, selon l'équation A.3, il suffirait de réduire au minimum V_{DD} pour diminuer de manière quadratique la consommation de puissance. Mais c'est sans compter sur le problème de la variation de la tension de seuil V_{th} exploitée dans le *Body Biasing* RBB-FBB, qui influence de manière exponentielle la consommation statique (voir equ. A.2).

Les modules de DVFS et de *Dynamique Body-Biasing* doivent trouver un compromis entre consommation dynamique P_{dyn} et consommation statique P_{stat}. De plus, un autre paramètre est à prendre en compte : la fréquence. Afin d'obtenir de bonne performances, les SoC doivent être capables de fonctionner à des fréquences suffisantes lorsque ceci est requit. Cependant, un mauvais ajustement de la fréquence et de l'alimentation peut entraîner des fautes temporelles qu'il faut également considérer [55].

Les paramètres intervenant dans la gestion du compromis performance/consommation en font un problème complexe d'optimisation.

A.4.3 Capacités de découplage

Outre les modules d'alimentation propres aux systèmes sur puce, l'ajout de capacités de découplage est indispensable à leur bon fonctionnement. Elles sont montées en parallèle et relient l'alimentation du SoC à la masse du circuit (voir Fig.A.14). Ces capacités agissent en filtres passe-bas qui limitent les amplitudes des variations de tension de brève durée. Elles permettent donc de stabiliser la tension d'alimentation en lissant les effets provoqués par les autres éléments du circuit : pics de consommation, brèves coupures d'alimentation.

A.5 Les bus

Comme nous l'avons expliqué dans le début de la Section 1.1, la réutilisation des IP-block est très courante pour la conception de nouveaux systèmes. Ceci implique que les nombreuses interfaces des différents blocs doivent être prévues à cet effet. C'est pourquoi les bus doivent permettre la réutilisation de divers IP d'une part, et d'autre part, être suffisamment performants pour avoir une fluidité dans le transfert des données [81].

Dans les SoC, ces différentes considérations sont traitées par les spécifications ARM AMBA® (*Advanced Microcontroller Bus Architecure*) [16, 127, 56]. La hiérarchie de cette

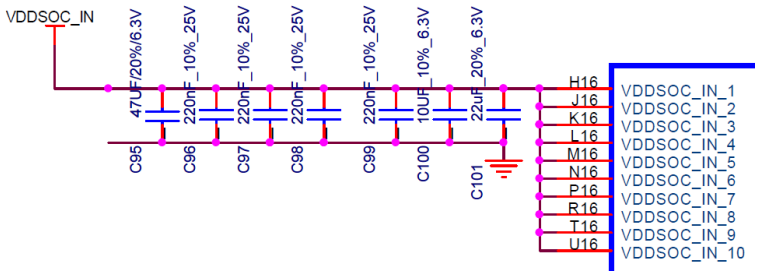


FIGURE A.14 – Condensateurs de découplage à l'entrée de l'alimentation d'un SoC

architecture se compose de 3 bus principaux : un bus central qui relie les différents éléments d'un système entre eux, des bus système qui connectent les éléments entre eux deux à deux et des bus périphériques qui relient des périphériques extérieurs au SoC. Ces bus sont interconnectés via des passerelles (*bridges*) qui servent d'interfaces entre les différents protocoles de communication. Ces passerelles sont des contrôleurs avec une mémoire tampon pour l'échange de données. Depuis 1996, cette architecture de bus a été améliorée, pour en 2003, avec la sortie du processeur ARM7TDMI, atteindre la version AMBA3 qui inclut un bus faisant partie intégrale de la solution de débogage. Cette architecture convient parfaitement aux systèmes embarqués complexes et a largement été utilisée dans les mobiles. Parce que trop complexe et inappropriée pour notre étude, nous ne ferons pas la description des spécifications de ces bus qui, aujourd'hui, en sont à leur 5$^{\text{ème}}$ version [5]. Néanmoins, on peut citer :

- AXI (Advanced Extensible Interface). Souvent utilisé comme « colonne vertébrale » d'un système. C'est un bus qui permet des transferts simultanés de données entre plusieurs modules travaillant à des fréquences différentes.
- AHB (Advanced High-performance Bus). Bus permettant des transferts haut débit entre deux modules travaillant à la même fréquence.
- APB (Advanced Peripheral Bus). Bus ayant des caractéristiques permettant de communiquer avec des périphériques.
- ATB (Advanced Trace Bus). Bus dédié pour le débogage. Il est implanté en parallèle des autres bus et permet d'observer les différents modules durant leurs fonctionnement.

La figure A.15 présente un exemple de ramification des bus AMBA®. Un des challenge qui se présente aux fabricants de SoC est l'interconnexion physique des différents bus avec les éléments intégrés dans les systèmes. Comme illustré par la figure A.16, ils utilisent des procédés faisant intervenir en moyenne une dizaine de couches superposées de métal/isolants reliant les différents modules du système [109]. Ces réseaux de bus électroniques de plus en plus élaborés relient les processeurs, les mémoires et les périphériques [53] (Chapitre "Interconnect").

5. https://www.arm.com/products/amba-open-specifications.php

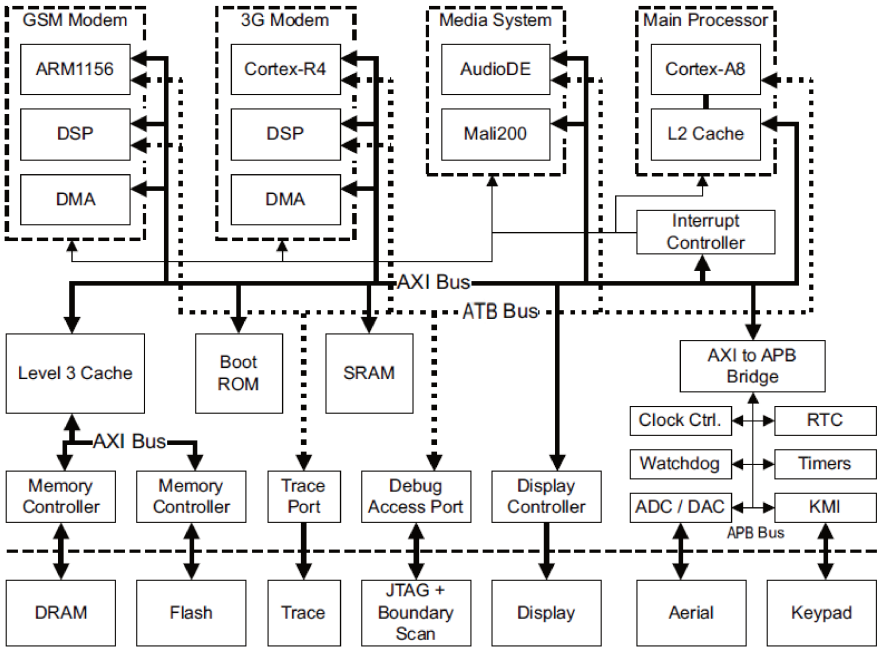


FIGURE A.15 – Schéma simplifié d'une architecture AMBA 3 d'un SoC de mobile (source [137])

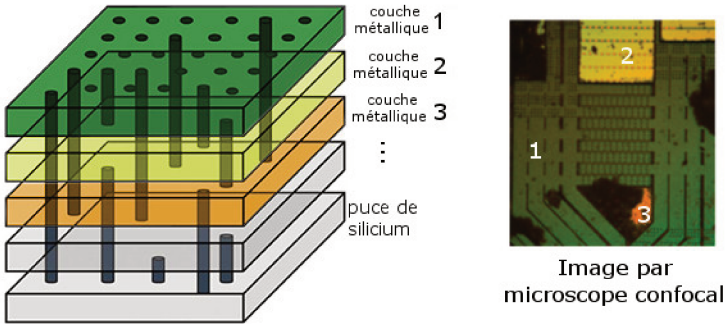


FIGURE A.16 – Différentes couches métalliques d'interconnexions relient les différents éléments du système entre eux.

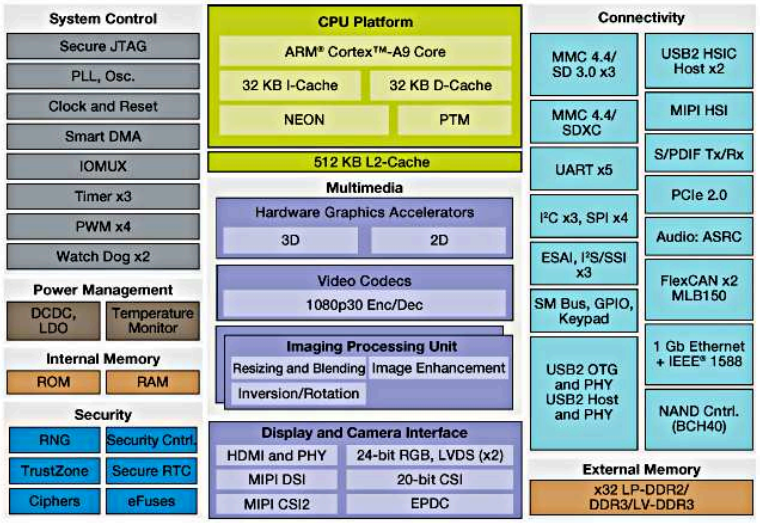


FIGURE A.17 – Schéma bloc simplifié des périphériques d'un SoC [6]

A.6 Les divers périphériques

Ici, le terme « périphériques » désigne des modules qui ajoutent des fonctionnalités au système principal. Dans ce paragraphe, nous présentons ceux qui sont caractéristiques aux systèmes sur puce. La figure A.17 met en lumière la diversité que l'on peut observer dans les périphériques des SoC d'aujourd'hui. Outre les processeurs, les mémoires, la gestion de l'alimentation, le débogage et la sécurité, on trouve presque systématiquement des périphériques liés au multimédia, aux interfaces avec l'extérieur et au système.

A.6.1 Périphériques liés au multimédia

Les systèmes complexes embarqués dans les appareils tels que les téléphones portables traitent des données du type multimédia. En effet, les téléphones d'aujourd'hui permettent de filmer, photographier, jouer et regarder des vidéos. Ceci implique que des modules spécialisés dans ces domaines soient présents pour assister le processeur. Le multimédia utilise des périphériques pour l'acquisition, le traitement et la diffusion des données. L'acquisition des données est réalisée par des contrôleurs gérant l'interface entre le SoC et les modules extérieurs, comme par exemple, les modules de caméra ou de microphone. Ensuite, le traitement de ces données est effectué par des processeurs et des accélérateurs matériels dédiés. Ils travaillent en parallèle avec le processeur principal et le déchargent des calculs de traitement graphique pour une meilleure fluidité d'exécution. Différents types de processeurs conçus pour le traitement rapide d'images (GPU) peuvent être implantés dans les SoC [103]. Enfin, la diffusion de données se fait au travers de modules tels que les haut-parleurs et les écrans, qui doivent être capable d'interpréter les informations émis par le SoC. D'autres contrôleurs périphériques sont dédiés à ces tâches.

6. http://www.nxp.com/products/microcontrollers-and-processors/arm-processors

A.6.2 Périphériques liés aux interfaces avec l'extérieur

Les SoC intégrés dans des objets connectés sont destinés à échanger des données avec l'extérieur. Que ce soit pour se connecter sur un réseau, lire des mémoires extérieures, ou allumer une LED, le système sur puce doit être capable de communiquer suivant les différents protocoles courants de communication. Pour cela, différents modules dédiés incluant les standards industriels tels que USB, FireWire, Ethernet, UART, SPI, etc. Le bloc *Connectivity* de la figure A.17 donne une liste non exhaustive des différents protocoles dont un SoC peut avoir besoin pour fonctionner.

A.6.3 Périphériques système

Ces périphériques ont pour objectif d'assister et de surveiller le bon fonctionnement de systèmes si compliqués. Parmi ceux-ci on peut citer les *timers* et les *watch dog*, minuteurs et chien de garde en français, qui sont des contrôleurs destinés à surveiller temporellement la bonne exécution d'un processus donné. Le *timer* permet de paramétrer la génération périodique de signaux (interruptions) sans l'intervention du CPU. Le *watch dog* génère une interruption lorsqu'une anomalie survient, *i.e.* lorsque le temps d'exécution d'un processus dépasse un maximum qui lui a été attribué. On peut également citer les multiplexeurs d'entrées/sorties (IOMUX). Les systèmes sur puce ont un nombre très élevé de signaux à gérer, bien supérieur au nombre de billes qu'ils ont pour les connecter au PCB. Par conséquent, l'IOMUX définit de manière dynamique la bille qui acheminera un signal donné. Enfin, parmi les périphériques remarquables, on peut citer les modules d'accès direct à la mémoire, aussi dénommés par le terme anglais *Direct Memory Access* (DMA). Les DMA permettent à certains sous-systèmes matériels d'accéder à la mémoire principale (RAM) sans l'intervention du processeur principal. Ils dispensent le CPU de traiter les opérations de transferts mémoires pour d'autres modules, évitant ainsi qu'il soit accaparé par ces transferts qui le rendraient indisponible pour exécuter d'autres tâches.

A.6.4 Convertisseurs AN et NA

Certains modules possèdent des interfaces analogiques comme, par exemple, les microphones et les haut-parleurs. Pour fonctionner, ils utilisent les convertisseurs analogique-numériques (AN) et numérique-analogiques (NA) du SoC pour échantillonner et émettre leurs signaux sonores.

Annexe B

Attaques side-channel classiques

Dans cette annexe, nous décrivons les principales attaques par canaux cachés appliquées aujourd'hui sur les systèmes électroniques effectuant des calculs cryptographiques.

Sommaire de l'annexe

L'attaque des systèmes électroniques par la mesure des variations de leurs grandeurs physiques a été initiée par les travaux de Kocher [78, 79]. Dans la communauté, ce type de cryptanalyse est appelée attaques par canaux auxiliaires ou *side-channel attacks* en anglais. Avec l'apparition de nouveaux algorithmes et l'exploitation de chemins d'attaques inédits, une multitude de méthodes ont été développées, chacune utilisant un modèle de fuite adaptée à l'implémentation du protocole cryptographique qu'elle cible.

B.1 Modèles de fuites d'information

Une majorité des différentes méthodes d'attaques SCA utilisent des modèles de fuites. Il s'agit de modèles mathématiques qui tentent de décrire le comportement de consommation des circuits en fonction des opérations effectuées par les algorithmes cryptographiques. Ils expriment une fuite d'information L et sont composés de deux parties : une déterministe $\delta(\cdot)$, dépendante du circuit et des opérations effectuées, et l'autre indépendante, considérée comme du bruit B. La partie déterministe fait intervenir une donnée corrélée avec la clé recherchée k. Cette donnée peut être le résultat d'une opération arithmétique avec une autre donnée connue x ou un transfert mémoire par exemple. Le bruit est modélisé par un bruit blanc gaussien. Ainsi, les fuites d'informations sont représentées par l'équation B.1 suivante :

$$L(x,k) = \delta(x,k) + B \tag{B.1}$$

Au cours de l'évolution des attaques SCA, les fonctions déterministes $\delta(\cdot)$ ont étés affinées. On trouve notamment les modèles suivants :

- **Le poids de Hamming** $H_W(\cdot)$. Le poids de Hamming est le nombre de bits à `1` dans une donnée D d'un processeur d'architecture N bits. En binaire, cette donnée est codée par $D = \sum_{i=0}^{N-1} d_i 2^i$ avec $d_i \in \{0,1\}$ et son poids de Hamming est simplement $H_W(D) = \sum_{i=0}^{N-1} d_i$. Ces modèles considèrent la valeur des bits contenus dans une donnée manipulée [78, 79].

- **La distance de Hamming** $H_D(\cdot)$. La distance de Hamming est le nombre de bits différents entre la donnée D_1 et D_2. Cela revient à compter tout les bits à `1` après avoir calculé l'opération OUexclusif entre D_1 et D_2. Sur un processeur N bits, cela est définit par la formule $H_D(D_1, D_2) = \sum_{i=0}^{N-1} d_{1,i} \oplus d_{2,i}$. Ces modèles considèrent la transition des bits [34]. Ils représentent le remplacement d'une valeur contenue dans un registre par une autre. Ils se rapprochent plus de la réalité physique des observations.

- **La régression linéaire** $LR(\cdot)$. Les modèles utilisés dans [122, 48] considèrent l'influence de la position de chaque bits sur la mesure physique. Pour cela, les auteurs font intervenir des coefficients α_i qui pondèrent les différents bits. La régression linéaire est utilisée pour déterminer la valeur de ces coefficients afin de minimiser la différence entre les observations physiques $L(\cdot)$ et le modèle théorique $\delta(\cdot)$. Ainsi, par exemple, une donné D pour une architecture N bits sera modélisée par $LR(D) = \alpha_{-1} + \sum_{i=0}^{N-1} \alpha_i \cdot d_i$. Ces modèles tendent à se rapprocher de la réalité des architectures matérielles dans

lesquelles les bus ont de grandes tailles et l'influence de chaque bit dépend de sa position.

Nous illustrons l'utilisation d'une partie de ces modèles de fuites à travers les attaques *side-channel* que nous décrivons dans cette section. Nous ne les citerons pas de manière exhaustive mais nous présenterons les plus répandues d'entre elles.

B.2 Simple Power Analysis (SPA)

La première et la plus simple des attaques *side-channel* est la *simple power analysis*, la simple analyse de puissance en français. Elle consiste en l'observation de la consommation de courant d'un circuit pour deviner la clef utilisée. Ainsi, en connaissant l'algorithme qui s'exécute et en utilisant ce procédé, un attaquant peut casser la sécurité théorique d'un protocole cryptographique. La visualisation de la consommation se fait grâce à un oscilloscope. Dans sa version d'origine [78], l'attaque SPA considérait la mesure de la tension aux bornes d'une résistance placée sur la masse du circuit. Cette attaque a ensuite été améliorée dans l'étude [114]. Dans celle-ci, les auteurs analysent, non plus la consommation, mais les rayonnements EM : *Simple Electro-Magnetic Analysis* (SEMA). Dans les deux versions, les échantillons acquis présentent des pics et des durées qui sont l'image des opérations effectuées par le processeur. Ici, on considère que c'est la valeur des bits qui influence le comportement des circuits. Par conséquent le modèle de fuite utilisé est le poids de Hamming.

Algorithm 2: Exponentiation modulaire du RSA

Input: la clé privée : $k = (k_{l-1}, ..., k_0)$, le message chiffré : c, le module : n
Output: le message en clair : m

```
1 n ← 1
2 for i ← 0 to l − 1 do
3 |   c ← c^2 mod n
4 |   if k_i = 1 then
5 |   |   m ← m · c mod n
6 |   end
7 end
```

La première SPA attaquait un protocole de chiffrement RSA. La partie contenant une faille sur l'information secrète est obtenue par le calcul d'exponentiation modulaire décrite dans l'algorithme 2. Cet algorithme effectue des opérations en fonction de la valeur du bit de clé considéré. Si celui-ci vaut `1`, alors une opération de multiplication modulaire est effectuée tandis que s'il vaut `0`, rien n'est fait. Ceci a une répercussion directe sur la consommation du circuit. Comme illustré par la Fig. B.1, les pics de consommation correspondent au calcul de la multiplication lorsque le bit vaut `1` et les « creux » représentent les `0`. Ainsi, en l'absence de protections ou contre-mesures, la valeur de la clef secrète utilisée peut être simplement lue de gauche à droite grâce à un oscilloscope.

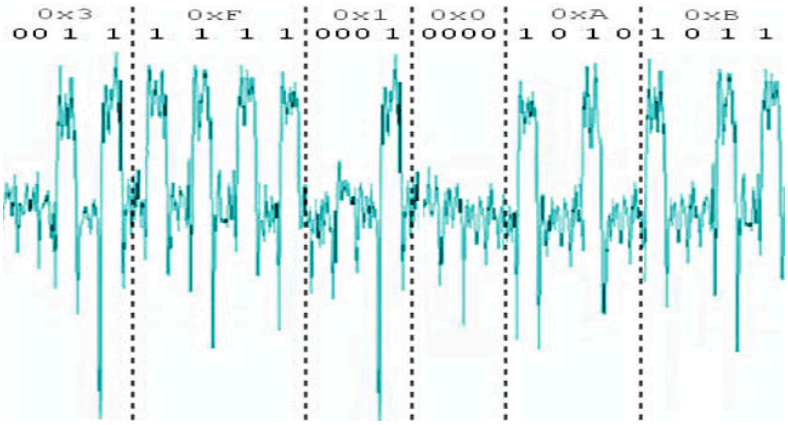


FIGURE B.1 – SPA de l'alimentation d'un protocole RSA sans contre-mesures

Aujourd'hui, la SPA n'est plus efficace sur les produits sécurisés. Les développeurs d'algorithmes cryptographiques sont conscients de la menace que représentent de telles attaques. Ils ont à leurs disposition une multitude de contre-mesures pour parer à ce genres d'éventualités. Les protections sont essentiellement des modifications apportées aux algorithmes. Par exemple, l'une d'entre elles consiste en l'ajout de calculs visant à équilibrer la consommation quoi qu'il arrive. Ainsi, lorsque le bit de la clé est nul, un calcul servant de leurre est effectué. Il en résulte une consommation non dépendante de la donnée manipulée.

Cependant, en considérant les avantages des champs EM, les attaques SEMA peuvent être utilisées dans les SoC pour effectuer une phase de rétro-ingénierie sur d'autres processus de sécurité. Nous verrons cela dans le $4^{\text{ème}}$ et le $5^{\text{ème}}$ chapitre de cette étude.

B.3 Differential Power Analysis (DPA)

L'attaque par différence des moyennes de consommation aussi connue sous son appellation anglaise, *Differential Power Analysis* a été introduite dans [79]. De la même manière que pour la SPA, une version utilisant la mesure des émissions électromagnétiques est envisageable. C'est cette dernière version qui sera plus adaptée aux SoC. Il s'agit d'une attaque plus sophistiquée que la précédente, faisant intervenir une phase de traitement statistique des données. Pour simplifier la recherche du secret, cette attaque applique la méthode du « diviser pour régner ». Autrement dit, elle ne considère pas directement toute la clé, mais travaille sur chacun de ses octets tour à tour. Par exemple, pour une chiffrement AES avec une clé de 16 octets, cette attaque considère progressivement chaque octet. Au lieu de chercher les 256^{16} valeurs de clés possibles, cette attaque s'attachera à trouver successivement les 256 valeurs de chaque octet. Cela revient à calculer 16×256 valeurs au lieu de 256^{16}, ce qui représente un temps de calcul bien plus raisonnable. Une attaque DPA se déroule suivant plusieurs étapes :

1. Une première attaque SPA qui permet de déterminer l'algorithme qui s'exécute sur le cryptosystème. Puisque ces algorithmes sont standardisés, il est relativement simple de les identifier par le nombre de répétitions de leurs motifs. Par exemple, les 16

répétitions du même motif sur figure B.2 permettent d'identifier les 16 rondes du calcul DES. De la même manière, les 10 rondes de l'AES, l'exponentiation modulaire du RSA et les autres algorithmes sont identifiables.

2. La sélection d'un endroit de l'algorithme où un bit b dépend de la valeur de l'octet d'indice p de la clé K. Il s'agit en général d'un bit impliqué dans une opération de OU-exclusif avec un octet K_p de la clef.

3. L'utilisation d'une fonction de sélection D qui permettra de déterminer la valeur du bit b choisit en fonction du message M ou du chiffré C et de l'hypothèse faite sur la valeur de l'octet $\widetilde{K_p}$ de la clé. Cette fonction de sélection ne prendra donc que deux valeurs : 0 ou 1.

4. Une phase d'échantillonnages dans laquelle un attaquant associe un ensemble de N messages M_i ou de N chiffrés C_i à leurs mesures de consommation de puissance. Ces mesures sont appelées des *traces* et sont notées $T_i[j]$ où $j = \{0, \ldots, t\}$ représente les t échantillons temporels de la trace. On note (M_i, C_i, T_i) l'association du i-ème message M_i avec son chiffré C_i et sa mesure de la consommation T_i durant le calcul de l'algorithme identifié dans l'étape 1.

5. Afin de pouvoir effectuer un traitement statistique, il est nécessaire que toutes les mesures de consommation soient alignées. Pour cela un prétraitement consiste à sélectionner une trace de référence T_0 et d'aligner toutes les autres par rapport à celle-ci.

6. L'avant-dernière phase consiste en le regroupement des différentes traces T_i en fonction de la valeur du bit b_i sélectionné. Cette valeur est issue de la fonction de sélection qui, dans cet exemple, dépend de la valeur du chiffré C_i et de l'hypothèse sur l'octet de clé $\widetilde{K_p}$: $D(C_i, \widetilde{K_p})$. Ainsi, la totalité des traces T_i, $i \in \{0, \ldots, N-1\}$ sont regroupées dans deux ensembles : $\Gamma_0 = \{T_i \mid D(C_i, \widetilde{K_p}) = 0\}$ ou $\Gamma_1 = \{T_i \mid D(C_i, \widetilde{K_p}) = 1\}$

La dernière étape consiste en le calcul de la courbe différentielle des moyennes $\Delta_{\widetilde{K_p}}$. Il s'agit du distingueur qui permettra de déterminer si une hypothèse est valide ou pas. Cette courbe s'obtient par le calcul de la différence de la moyenne des traces contenues dans l'ensemble Γ_0 avec celles contenues dans l'ensemble Γ_1 :

$$\Delta_{\widetilde{K_p}}[j] = \frac{\sum_{i=1}^{N} D(C_i, \widetilde{K_p}) \cdot T_i[j]}{\sum_{i=1}^{N} D(C_n, \widetilde{K_p})} - \frac{\sum_{i=1}^{N} (1 - D(C_i, \widetilde{K_p})) \cdot T_i[j]}{\sum_{i=1}^{N} (1 - D(C_i, \widetilde{K_p}))} \tag{B.2}$$

Si l'hypothèse sur la valeur de l'octet de clé $\widetilde{K_p}$ est fausse, alors la valeur du bit b_i donnée par la fonction de sélection $D(C_i, \widetilde{K_p})$ le sera également. Les traces T_i seront rangées aléatoirement, *i.e.* suivant aucune règle dans les ensembles Γ_0 et Γ_1. Par conséquent, la différence des moyennes de ces deux ensembles $\Delta_{\widetilde{K_p}}$ ne représente que du bruit. En revanche, si l'hypothèse $\widetilde{K_p}$ vérifie la valeur de l'octet de clé secrète K_p, alors les ensembles Γ_0 et Γ_1 contiennent des mesures de consommation rangées en fonction de la valeur du bit sélectionné. Dans ce cas, $\Delta_{\widetilde{K_p}}$ fait ressortir un pic qui représente la variation de consommation liée à l'utilisation de ce bit. Remarquons que cette information dépend de la valeur d'un

bit, et donc comme pour la SPA, c'est le poids de Hamming qui est utilisé comme modèle de fuite. La figure B.3 illustre le calcul de la différence des moyennes pour une hypothèse d'octet fausse et une hypothèse correcte. En répétant ces opérations pour les N octets $K_{0 \leq p \leq N-1}$, un attaquant est à même de retrouver la valeur de la clef secrète.

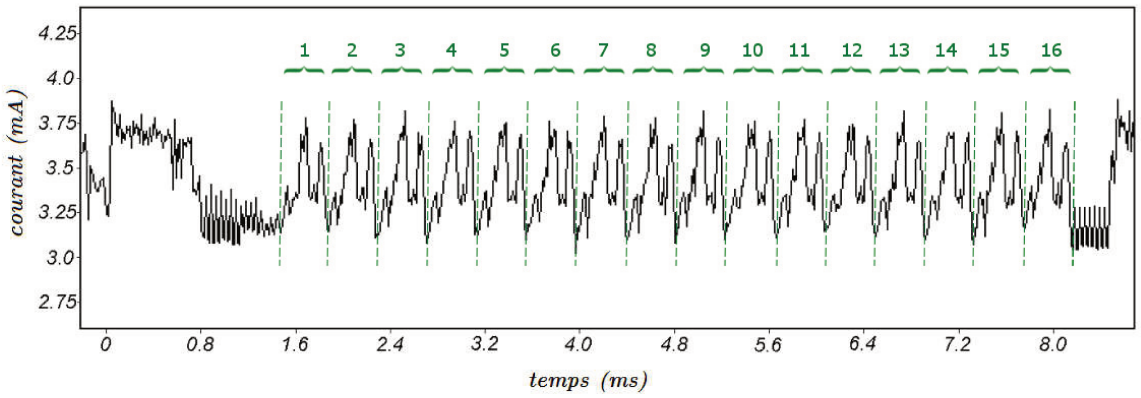


FIGURE B.2 – SPA de l'alimentation d'un protocole DES

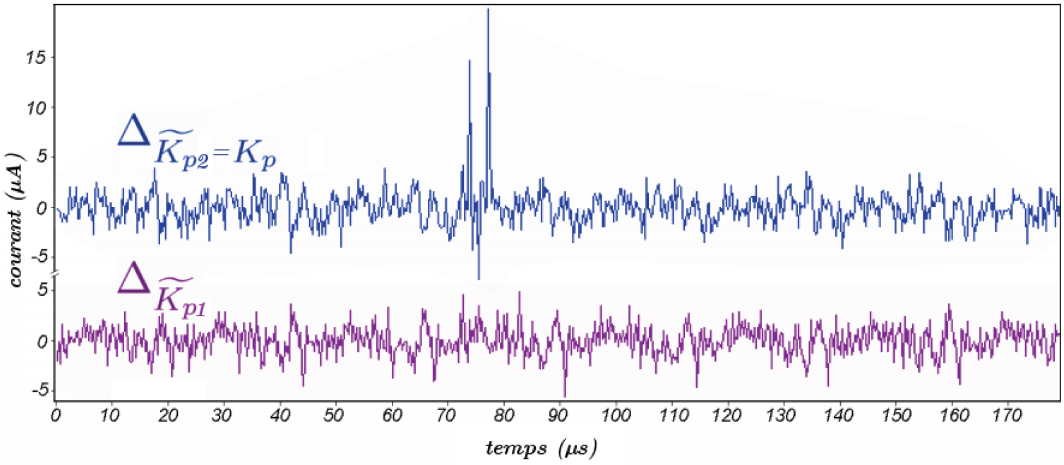


FIGURE B.3 – Calculs de la différences des moyennes $\Delta_{\widetilde{K_p}}$ pour une hypothèse d'octet de clé incorrecte $\widetilde{K_{p1}}$ et une hypothèse correcte $\widetilde{K_{p2}} = K_p$

B.4 Correlation Power Analysis (CPA)

L'attaque par analyse de l'alimentation par corrélation a été proposée dans *Correlation Power Analysis* [34]. La méthodologie de cette attaque est similaire à celle de la DPA en y ajoutant toutefois des améliorations sur le modèle de fuite. Dans cette étude, les auteurs supposent que la consommation dépend, non pas de la valeur des bits, mais plutôt de leur

transition. Ils supposent également qu'elle est la même lorsqu'un bit passe de `0` $\rightarrow$ `1` ou `1` $\rightarrow$ `0`. Par conséquent, pour décrire la consommation W en fonction de la valeur de la clé K, ils utilisent un modèle de fuite linéaire basé sur la distance de Hamming H_D (voir section B.1) :

$$W = a \cdot H_D(M, K) + B \tag{B.3}$$

où a est un gain et M une donnée connue et contrôlée. Dans cette attaque, le distingueur utilisé est le coefficient de corrélation linéaire de Bravais-Pearson :

$$\Delta_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)}{\sqrt{\mathbb{E}(X^2) - \mathbb{E}(X)^2}\sqrt{\mathbb{E}(Y^2) - \mathbb{E}(Y)^2}} \tag{B.4}$$

Ce coefficient traduit l'existence d'une relation linéaire entre les variables X et Y et est compris entre -1 et 1. Plus ce coefficient est proche de ses extrêmes, plus il existe une relation linéaire qui permet d'exprimer une variable en fonction de l'autre. A l'inverse, $\Delta_{X,Y} = 0$ signifie qu'il n'existe pas de relation linéaire reliant X à Y.

Le principe de cette attaque est de calculer la corrélation linéaire entre les mesures de consommation de puissance et les données manipulées. Comme pour la DPA, on applique la méthode du diviser pour régner afin de réduire le nombre de données à traiter. Souvent la taille des données considérées est l'octet. Pour la CPA, il est aussi nécessaire pour un attaquant de construire un ensemble qui relie la valeur des données manipulées aux mesures physiques de la consommation (voir l'étape 4 de la DPA). Soit $(M_i, T_i[j])_{0 \leq i \leq N-1}$ l'ensemble qui associe le message M_i utilisé lors du calcul cryptographique avec sa trace mesurée $T_i[j]$. On note $H_{i,\widetilde{K}}$ la distance de Hamming entre le message M_i et $\widetilde{K}$, une hypothèse sur la valeur de l'octet de la clé secrète : $H_{i,\widetilde{K}} = H_D(M_i, \widetilde{K})$. Pour l'hypothèse $\widetilde{K}$, le calcul du distingueur donné par l'équation B.4 s'exprime alors :

$$\Delta_{T,\widetilde{K}}[j] = \frac{N \sum_{i=0}^{N-1} T_i[j]\, H_{i,\widetilde{K}} - \sum_{i=0}^{N-1} T_i[j] \sum_{i=0}^{N-1} H_{i,\widetilde{K}}}{\sqrt{N \sum_{i=0}^{N-1} T_i^2[j] - \left(\sum_{i=0}^{N-1} T_i[j]\right)^2}\sqrt{N \sum_{i=0}^{N-1} H_{i,\widetilde{K}}^2 - \left(\sum_{i=0}^{N-1} H_{i,\widetilde{K}}\right)^2}} \tag{B.5}$$

Une mauvaise hypothèse d'octet de clé $\widetilde{K}$ ne permet pas au coefficient de corrélation de mettre en valeur une relation linéaire entre les traces $T_i[j]$ et les données $H_{i,\widetilde{K}}$. La corrélation est faible, *i.e.* $|\Delta_{T,\widetilde{K}}[j]| \ll 1$. En revanche si l'hypothèse $\widetilde{K}$ vérifie la valeur de l'octet de clé utilisé pour l'opération cryptographique, alors une forte corrélation peut être observée, *i.e.* $\widetilde{K} = K \Rightarrow |\Delta_{T,\widetilde{K}}[j]| \rightarrow 1$. En calculant $\Delta_{T,\widetilde{K}}[j]$ pour toutes les valeurs possibles de $\widetilde{K}$, il est possible d'identifier la valeur qui maximise la corrélation entre les données manipulées et les traces mesurées.

B.5 Méthodes Supervisées

Une dernière catégorie d'attaques par canal auxiliaire utilisant l'augmentation de la puissance de calcul des ordinateurs est apparue après les années 2000. Il s'agit d'attaques

se basant sur le *Machine Learning*, l'apprentissage automatisé en français. Cette catégorie regroupe les *Template Attacks* [38], les machines à vecteurs supports [43], les réseaux de neurones et plus généralement, les méthodes d'apprentissage automatique. La particularité de cette famille d'attaques est de se passer de modèle de fuite. En effet, l'application d'algorithmes d'apprentissage sur des données ne nécessite pas d'avoir de préjugés sur celles-ci, ce qui est un atout. Cependant, l'une des principales difficultés est de considérer des données discernables et classables entre elles. L'autre contrainte est qu'un attaquant utilisant les méthodes d'apprentissage automatisé doit disposer d'une copie du système qu'il cible. De plus, il doit pouvoir contrôler l'ensemble des paramètres de cette copie, en particulier les valeurs du message et de la clé utilisés pour les opérations cryptographiques. Toutes ces attaques se déroulent en deux phases : un apprentissage et une exploitation. Durant l'apprentissage, l'attaquant construit un modèle mathématique sur sa copie afin de prédire au mieux les valeurs des données manipulées en fonction des traces de consommations mesurées. Après quoi, le système attaqué est confronté à ce modèle. En reproduisant une partie des manipulations effectuées durant l'entraînement, l'attaquant essaye de faire prédire à son modèle la valeur secrète de la donnée manipulée en mesurant la consommation. Ce type d'attaques est possible uniquement dans un contexte bien précis et demande de lourds calculs. Cependant, une fois un modèle de prédiction construit, il est possible d'attaquer un système en peu de manipulations. Actuellement, c'est le sujet le plus innovant dans le milieu de la cryptanalyse et de nouvelles techniques sont régulièrement proposées [86].

Annexe C

Imagerie microscopique

Cette annexe décrit les principales techniques d'imageries qui peuvent être utilisées pour de la rétroconception de systèmes électroniques.

Sommaire de l'annexe

L'imagerie microscopique est une technique d'observation qui permet d'obtenir des informations visuelles vis-à-vis d'un circuit électronique. En fonction de la technologie utilisée, diverses données peuvent être obtenues comme par exemple la localisation spatiale ou l'identification de l'instant de fonctionnement d'un bloc IP. L'imagerie microscopique est un puissant outil de rétroconception qui apporte le complément d'information nécessaire à un attaquant pour pouvoir détourner un système. Sans entrer dans les détails ni être exhaustif, nous allons brièvement présenter les principaux types de microscopes utilisés sur des circuits intégrés.

C.1 Microscopie optique

Les microscopes optiques à lumière visible utilisent un système de lentilles pour agrandir l'image d'un échantillon observé. Plusieurs technologies existent et se différencient par notamment leur système d'éclairage. Pour procéder à l'observation des circuits, une certaine préparation de celui-ci est nécessaire, *i.e.* il faut enlever tous les boitiers et les protections afin que rien n'occulte les parties à observer (voir Fig. 1.5.a). Ceci est une manipulation invasive qui peut parfois être une contrainte (ex. peu d'échantillons). De plus, ces microscopes sont limités quant à leur résolution par le phénomène de diffraction de la lumière. En effet, outre les imperfections des éléments optiques, la résolution maximale d'une image agrandie par ce procédé est bornée par la limite de diffraction d'Abbe donnée par équation C.1.

$$d = \frac{\lambda}{2.NA} \qquad ; \qquad NA = n \sin\frac{\theta}{2} \tag{C.1}$$

Celle-ci définit la distance limite d en dessous de laquelle deux points ne se distinguent plus lorsqu'ils sont observés avec une lumière de longueur d'onde λ. L'ouverture optique NA, est fonction de l'angle du cône d'observation θ et de l'indice de réfraction du milieu n. En considérant que le spectre des longueurs d'ondes du visible S_λ est compris entre 380 nm $\leq S_\lambda \leq$ 780 nm et que l'ouverture optique des meilleures techniques actuelles est proche de 1,4 - 1,6, on peut calculer la distance minimale observable par un microscope optique. Celle-ci est encore supérieure à la valeur nécessaire pour observer des portes logiques utilisées dans un SoC. Pour illustrer cela, considérons une manipulation dans laquelle nous utilisons du matériel optique avec $NA = 1$ et une lumière visible avec une longueur d'onde faible : la lumière violette ($\lambda_v = 400$ nm). La distance minimale observable dans cette configuration sera $d = 200$ nm. Celle-ci est suffisante pour observer des circuits imprimés de quelques micromètres mais encore trop élevée pour observer des systèmes complexes embarqués gravés avec une technologie de l'ordre du nanomètre (voir le tableau A.2).

Malgré cette limite, certains microscopes à lumière visible fournissent des images de qualité avec des informations supplémentaires. C'est le cas, par exemple, de la microscopie confocale.

C.2 Microscopie confocale

La microscopie confocale est une amélioration de la microscopie optique. Bien qu'il faille toujours préparer un circuit imprimé de manière invasive pour pourvoir l'observer, cette technique offre des avantages au niveau de la netteté et de la perception du relief. En

effet, avec un microscope optique classique, lorsque l'on observe un échantillon qui possède une certaine profondeur, la mise au point ne peut se faire que dans le plan focal, *i.e.* que pour une profondeur donnée. Ceci provoque des zones de flou sur l'ensemble de l'image. Pour palier cet inconvénient, un microscope confocal éclaire le circuit avec un faisceau laser. En positionnant le plan focal de l'objectif à différentes profondeurs, l'échantillon est balayé point par point. Ceci permet de conserver uniquement les photons provenant du plan focal observé et d'éliminer les autres. L'ensemble des images ainsi constituées permet de créer informatiquement une représentation en relief de l'échantillon avec une résolution légèrement supérieure.

Plus coûteuse que la microscopie optique classique, cette technologie est essentiellement utilisée par les laboratoires pour effectuer de la rétro-ingénierie et distinguer les différents niveaux de couches présents sur un circuit intégré [133]. La figure 1.5.b montre le type d'image que l'on peut obtenir sur un SoC. Les différentes couleurs représentent les diverses couches observables. Cependant, comme pour la microscopie optique classique, cette méthode ne permet pas d'observer l'ensemble des couches décrites dans la section A.5. Elle donne uniquement des informations sur les 3 ou 4 premiers niveaux d'un SoC. Afin d'avoir davantage d'informations de manière non invasive, d'autres méthodes sont à utiliser.

C.3 Mesure des rayonnements infrarouges (IR)

La microscopie infrarouge utilise comme source lumineuse des ondes électromagnétiques de longueur d'onde infrarouges. Ces dernières possèdent la propriété de traverser le silicium. Ainsi, cette technique permet d'observer de manière non-intrusive les différentes couches et contacts d'un circuit sans avoir à enlever les couches supérieures. Elle permet également de localiser les zones actives d'un circuit au travers du substrat par photo-émission [52]. La résolution maximale est bornée par la longueur d'onde utilisée (voir la limite de diffraction donnée par l'équation C.1), ce qui, dans le cas des infrarouges, est toujours supérieure à la taille des technologies de gravures des processeurs actuels (voir le tableau A.2 de l'annexe A). Cette microscopie est principalement utilisée dans l'industrie et les laboratoires de rétroconception.

C.4 Microscopie par rayons X

La microscopie à rayons X fait partie des techniques d'observation non invasives qui permettent de voir les différentes couches présentes sur un SoC. Elle utilise des rayons électromagnétiques dont les longueurs d'ondes sont comprises entre 10^{-3} et 10 nm, et sous lesquels certains matériaux sont transparents, alors qu'à la lumière visible ils ne l'étaient pas. Cette propriété a rapidement été exploitée dans la rétroconception de circuits intégrés (voir figure 1.5.c). En effet, pouvoir observer les différents éléments présents dans un circuit imprimé ou dans un microprocesseur sans les détériorer est un avantage non négligeable. Bien que les circuits soient constitués par de plus en plus de couches métalliques et que leur finesse de gravure augmente, les performances des images obtenues avec la microscopie par rayon X progressent également. C'est le cas par exemple de l'impressionnant travail mené par des chercheurs du Paul Scherrer Institut [68] qui proposent le rendu 3D avec une résolution de 14,6 nm d'un microprocesseur Intel G3260 de technologie de gravure 22nm.

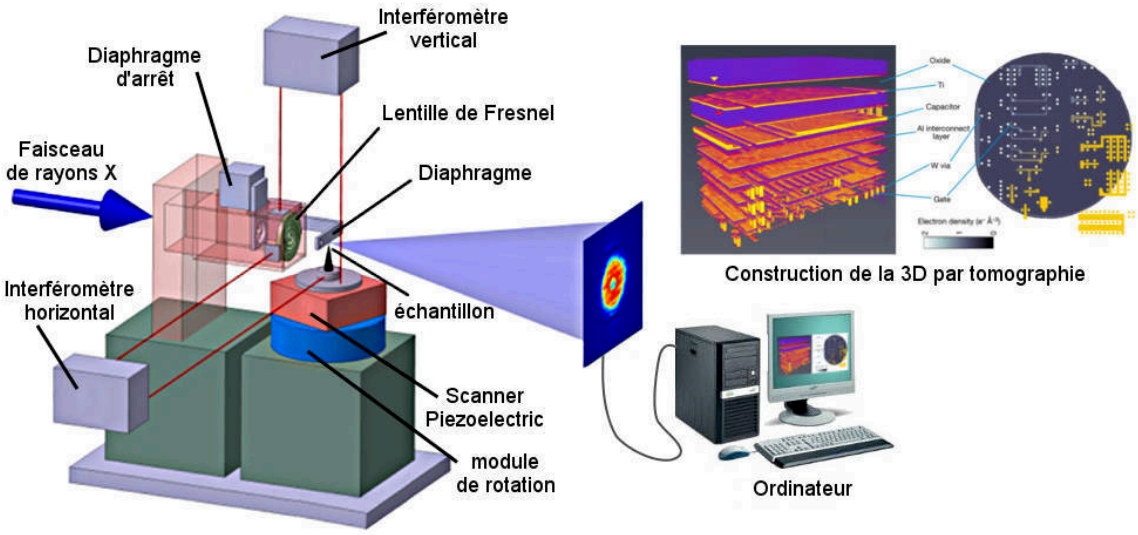


FIGURE C.1 – Microscope à rayons X utilisé dans [68]

Pour cela, ils utilisent l'imagerie ptychographique par rayons X à l'échelle du nanomètre pour générer l'ensemble des données nécessaires pour la tomographie (voir schéma Fig.C.1). A partir des images 2D transverses, ils reconstruisent informatiquement un modèle 3D [7].

Bien que cette technologie offre de nombreuses possibilités, elle nécessite une certaine expertise et des équipements onéreux qui, pour l'instant, sont réservés aux laboratoires spécialisés.

C.5 Microscopie électronique

Ce type de microscopie utilise les différentes propriétés d'interactions des électrons avec la matière des échantillons pour en extraire des informations. Ces interactions sont déterminées par la tension d'accélération des électrons, par le courant, par l'angle d'incidence du faisceau et par la composition des échantillons. Elles se manifestent sous la forme d'une perte d'énergie qui peut se traduire par diverses émissions (chaleur, électrons secondaires, électrons rétro-diffusés, rayons X, etc. voir Fig. C.2). Ces signaux transportent des informations caractéristiques de l'échantillon et peuvent être exploités. Par exemple, la composition chimique d'un circuit imprimé peut être obtenue par l'analyse dispersive en énergie des rayons X. Il existe une multitude de microscopes électroniques exploitant ces différents signaux, nous n'en citons qu'une infime partie appliquée à la rétroconception relative aux circuit intégrés [142] :

- La microscopie électronique à balayage (MEB - SEM). Se pratique à l'aide d'un type de microscope qui accélère et bombarde des électrons sur un échantillon à observer

7. Vidéo d'un modèle 3D : https://youtu.be/7VcgEHQrYao
8. Source : https://commons.wikimedia.org/w/index.php?curid=44224781

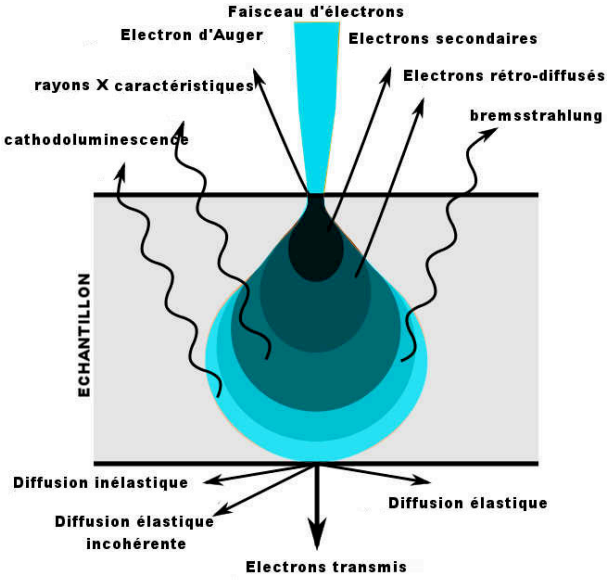


FIGURE C.2 – Diagramme d'interaction de la matière avec un flux d'électrons haute énergie (image adaptée [8])

avec une énergie cinétique de l'ordre du keV [149]. La surface de l'échantillon doit être au préalable préparée et nettoyée. Durant le bombardement, divers capteurs mesurent notamment les électrons secondaires et les électrons rétrodiffusés. Les premiers fournissent une information topographique de la surface de l'échantillon tandis que les seconds procurent une information sur la composition chimique. L'image de l'échantillon est construite informatiquement en assemblant les informations obtenues après avoir balayé l'ensemble de la surface avec ce procédé. La résolution d'un SEM est comprise approximativement entre 10 et 1 nm.

- La microscopie électronique en transmission (MET - TEM). De la même manière que les SEM, les microscopes utilisés bombardent un échantillon très fin avec des électrons, mais qui cette fois-ci, possèdent une énergie cinétique de l'ordre de la centaine de keV [106]. Ainsi, les électrons traversent l'échantillon et sont récupérés. Les interactions mesurées à partir des électrons transmis sont traitées de manière informatique pour reconstruire une image de l'échantillon. Cette technique permet d'obtenir des images d'une très haute résolution (0.1 nm), mais, en contrepartie, elle demande une lourde préparation des échantillons. Seules des couches très minces peuvent être observées. Pour cela des usinages très précis sont nécessaires. Cette phase complexe détermine la qualité des résultats.

- La microscopie en champ proche : *Scanning Capacitance Microscopy* (SCM) est un type de microscopie faisant intervenir une sonde électrode-aiguille qui balaye la surface en frôlant l'échantillon. La caractérisation de cette surface est faite à partir des mesures des changements de capacitance électrostatique entre l'électrode et l'échantillon. Pour cela une préparation minutieuse est nécessaire. La surface doit être

propre, dégagée et polie. Dans l'industrie des semi-conducteurs, ce type de microscopie est utilisée pour la caractérisation 2D en haute résolution de diélectriques minces [76, 83] ou plus généralement pour l'analyse de défauts [90].

Les microscopes électroniques rendent possible la mesure d'échantillons avec une très haute résolution. Celle-ci est augmentée par le fait que les longueurs d'ondes utilisées pour effectuer les mesures sont bien plus courtes que celles de la lumière visible (voir Equ. C.1). Cependant ce type de technologie est coûteuse et demande une certaine expertise ce qui restreint son utilisation à certains laboratoires spécialisés.

C.6 Comparaison des technologies

La comparaison de leurs principales caractéristiques est suffisante pour expliquer les capacités d'observation qu'il est possible d'obtenir par la rétro-ingénierie en fonction du contexte. Le tableau C.1 synthétise cette comparaison. Remarquons que, dans l'ensemble, ces techniques ne sont accessibles qu'à des laboratoires spécialisés. Dans la plupart des cas pour lesquels la résolution d'observation est suffisante pour observer les portes logiques d'un SoC, il est nécessaire de préparer l'échantillon de manière invasive. Cette préparation utilise un outillage spécifique et nécessite un savoir-faire particulier [26].

Microscope	**Optique (confocal)**	**Rayon X**	**SEM**	**TEM**	**SCM**	**IR**
Résolution spatiale	200 - 300 nm	$\leq$30 nm	1 - 10 nm	0.1 nm	$\leq$10 nm	3 - 30 μm
Profondeur de mesure	Dépend de l'indice de réfraction	0.1 - 50 mm	$\leq$ 10 nm	$\leq$ 100 nm	$\leq$ 15 nm	Silicium transparent dans les infrarouges
Préparation	invasif	non invasif	invasif	invasif	invasif	non invasif
Type de matériaux	Isolant, semiconducteur et conducteur	Isolant, semiconducteur et conducteur	Zone de conduction requise	Zone de conduction requise	Zone de conduction requise	Isolant, semiconducteur et conducteur

Tableau C.1 – Différentes technologies d'imagerie microscopique

Annexe D

Rappel des principes de fonctionnement du JTAG et SWD

Cette annexe rappelle brièvement le fonctionnement des deux mécanismes de débogage matériel les plus répandus dans les circuits intégrés : le JTAG et le SWD. Les détails de ces protocoles sont donnés dans les documents de référence [6, 70]

Sommaire de l'annexe

D.1 Le JTAG

La norme IEEE 1149.1, intitulée « Standard Test Access Port and Boundary-Scan Architecture »[6] définit un protocole et une architecture qui permettent de diagnostiquer et corriger des défaillances dans les systèmes électroniques via un unique connecteur : le port d'accès aux tests (*Test Access Port* : TAP). Trois fonctionnalités principales sont proposées :

- Le *Boundary-Scan* ou scrutation des frontières en français, est utilisé par les outils de test physique pour tester les courts-circuits et la continuité des pistes d'un appareil électronique. Chaque signal primaire d'entrée et de sortie est complété avec un élément mémoire appelé cellule Boundary-Scan. Ces cellules sont chaînées entre elles pour former le registre Boundary-Scan. Connaissant le schéma électrique du circuit, on applique des signaux sur les broches d'entrée du circuit, puis on relève les signaux des broches de sortie afin de s'assurer de la bonne qualité des pistes du circuit imprimé et des soudures. Pour effectuer ces test, les fabricants des systèmes conformes à la norme IEEE 1149.1, doivent fournir un fichier en langage *Boundary Scan Language Description* (BSDL) qui contient les informations sur la mise en œuvre du *Boundary-Scan* sur les composants compatibles de la carte électronique.

- Le *In-Circuit Debugger* ou *In-Circuit Emulator*, débogage interne au circuit en français, est prévue pour tester les fonctionnalités d'un circuit. En appliquant un ensemble de signaux logiques (appelé vecteur de test) sur les broches d'entrée du circuit, on stimule diverses fonctionnalités, puis on relève les niveaux logiques sur les broches de sortie pour s'assurer qu'ils correspondent aux valeurs attendues.

- Le *Debug Access*, l'accès au débogage en français, est utilisé par les outils de débogage matériel pour accéder aux ressources et aux fonctionnalité internes des puces électroniques. Cette fonctionnalité permet d'accéder aux routines internes des puces rendant accessibles et modifiables leurs ressources et opérations. Elle est utilisée par les outils de débogage logiciel pour vérifier le bon fonctionnement logique des systèmes en accédant par exemple aux registres et aux mémoires. Cette fonctionnalité est également mise en œuvre afin de charger les micrologiciels (*firmware*) dans les systèmes.

Bien que les principes du JTAG soient génériques à tous les types de SoC, les fonctionnalités implémentées sont propres à chaque fabricant, voire à chaque système électronique.

D.1.1 L'architecture Boudary-Scan

La norme IEEE 1149.1 a été élaborée pour définir une technique de conception des circuits électroniques en vue d'effectuer des tests de fonctionnement. Chaque signal primaire d'entrée et de sortie est complété avec un élément de mémoire dénommé cellule de *Boudary-Scan* (BSC). Les BSC sont chaînées entre elles pour former le registre *Boudary-Scan* (BSR) (voir Fig. D.1).

9. Adapté de : [105]

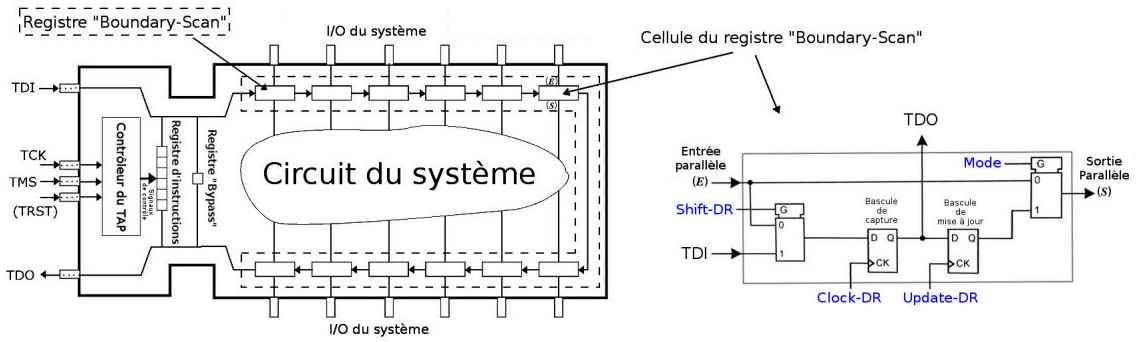


FIGURE D.1 – L'architecture *Boudary-Scan*[9].

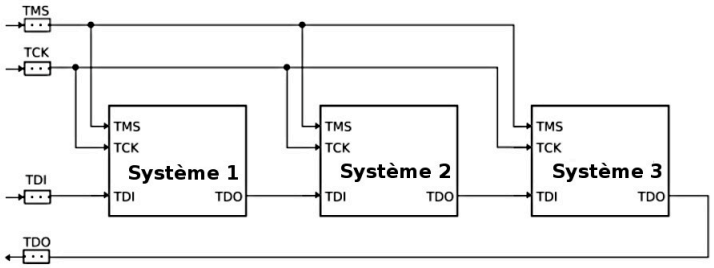


FIGURE D.2 – Chaînage des composants à tester, grâce à l'architecture *Boudary-Scan*

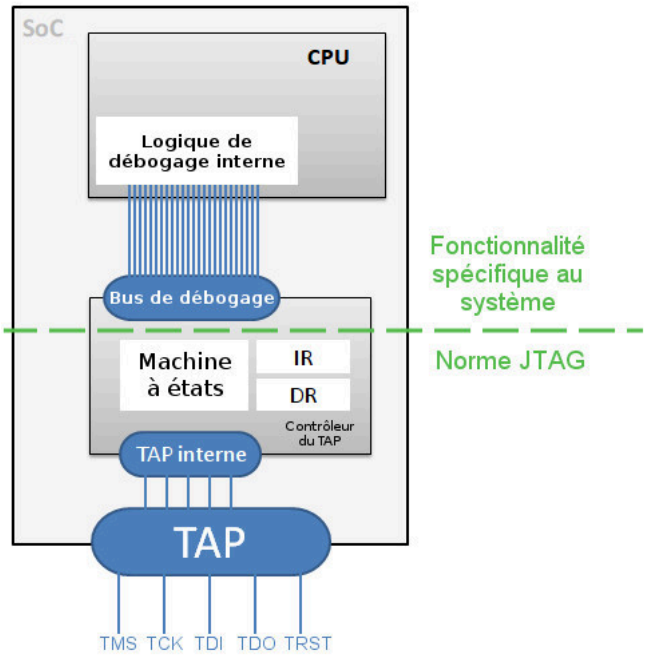


FIGURE D.3 – Le *Test Access Port*

Ce schéma de conception permet de connecter les différents composants entre eux de manière à former une « chaîne de scan » qui permet de tester ces derniers en une seule routine de débogage (voir Fig. D.2).

D.1.2 Le Test Access Port (TAP)

Le JTAG est définit par un protocole de communication série et une machine à état, accessible via le TAP. Le contôleur du TAP, aussi appelé contrôleur JTAG, est un élément passif qui ne répond qu'aux requêtes envoyées par l'outil de débogage via le connecteur TAP. La figure D.3 donne le schéma générique du *Test Access Port*. Cette dernière décrit :

- Le **TAP**, est la connexion physique entre l'outil de débogage externe et le système. Il s'agit de l'interface par laquelle transitent les signaux du protocole JTAG (voir Section D.1.3).
- La **machine à état**, aussi appelée « la machine à état JTAG », donne le mode de fonctionnement du contrôleur TAP. La norme JTAG définit 16 états, chacun associé à un mode opératoire (voir Section D.1.4).
- Un registre d'instruction **IR**, et plusieurs registres de données, **DR** (voir Section D.1.5).
- Le **Bus de débogage**, établissant la communication avec les logiques de débogage internes (voir Section D.1.6).

Les sessions de débogage sont effectuées par des outils appropriés qui écrivent ou en lisant des instructions ou des données dans les registres **IR** et **DR**.

D.1.3 Les signaux

La norme IEEE 1149.1 définit quatre signaux pour la communication série et le contrôle de la machine à état du JTAG.

- **TDI**, *Test Data In* (données en entrée de test), est un signal de données séries envoyé par la sonde de débogage vers le circuit à tester.
- **TDO**, *Test Data Out* (données en sortie de test), est un signal de données séries envoyé par le circuit testé, vers la sonde de débogage.
- **TMS**, *Test Mode Select* (sélection du mode de test), est un signal de configuration des états de la machine à état du TAP. Il est commun à tous les éléments de la chaîne de scan.
- **TCK**, *Test Clock* (horloge de test), est un signal externe d'horloge donné par la sonde de débogage. Comme TMS, l'horloge est commune à tous les éléments de la chaîne de scan. Les données envoyées par la sonde sur les ports TDI et TMS sont considérées au front montant de l'horloge, tandis que les données du port TDO sont mises à jour au front descendant.
- **TRST**, *Test Reset* (réinitialisation du test), est un signal optionnel qui réinitialise la machine à état du JTAG et les différents registres de la chaîne de scan.

D.1.4 La machine à état

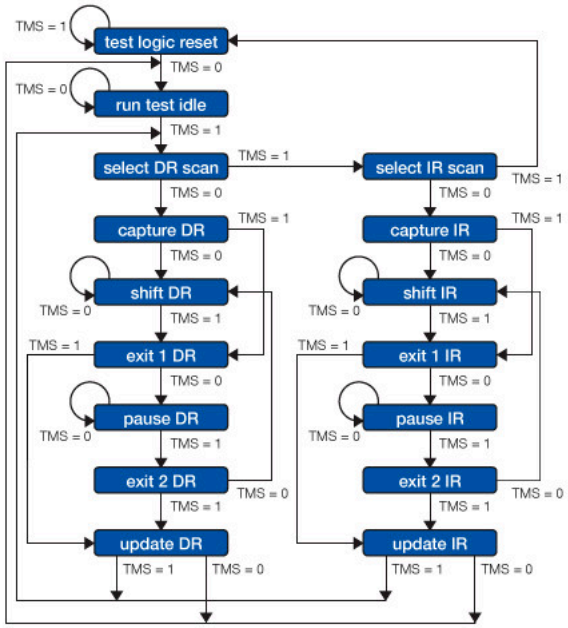


FIGURE D.4 – Machine à état du *Test Access Port*[10]

La machine à état définie dans la norme JTAG est illustrée figure D.4. Les 16 états de cette machine peuvent être atteints depuis n'importe quel autre état en envoyant une séquence de bits appropriée via le signal TMS. Parmi les différents états, on a en particulier :

- **Test logic Reset**. Cet état place le registre d'instruction dans sa valeur par défaut (**BYPASS** ou **IDCODE**, voir Section D.1.5). Il peut être atteint depuis n'importe quel autre état en alternant cinq "1" sur le signal TMS.

  Avec cet état, certains systèmes intègrent également une réinitialisation des registres du contrôleur JTAG et de la logique de débogage. Lorsque c'est le cas et qu'un programme de débogage atteint cet état, la connexion avec l'outil extérieur est perdue.

- **Run-test/Idle** et **Select DR-Scan** sont utilisés par la plupart des sondes de débogage comme un état d'attente entre deux opérations de décalage des données dans les registres.

- **Shift-IR**. Cet état permet à l'outil de débogage de charger une instruction dans le registre d'instruction. Le chargement est effectué bit par bit depuis le signal TDI. L'instruction est prise en compte lorsque l'état **Update-IR** est atteint.

- **Shift-DR**. Cet état permet à l'outil de débogage de récupérer bit par bit sur le signal TDO, les données retournées par un composant dans le registre de données.

10. Source :https://atbzdotorg.wordpress.com/2013/03/31/jtag-join-test-action-group-ieee-1149-1/

D.1.5 Registres d'instructions et de données

Le contrôleur JTAG comporte un registre principal d'instructions et des registres de données. Ces derniers sont au minimum deux : le registre **BYPASS** et le registre **BOUNDARY-SCAN** (voir Fig. D.5). La machine à état permet de mettre le contrôleur dans l'état de chargement du registre d'instruction afin de définir, à l'aide d'un décodeur, quel registre de données sera accédé. Peu de codes d'instruction sont définis par la norme JTAG.

- L'instruction **BYPASS**, utilisée lorsque la chaîne de scan est composée de plusieurs éléments.

- L'instruction **IDCODE**, n'est pas obligatoire mais souvent implémentée pour identifier les composants du système.

- Une de ces deux instructions est prise comme valeur par défaut pour le registre d'instruction.

Les tailles des registres d'instructions et de données ne sont pas spécifiées dans la norme IEEE. Cependant la taille des registres d'instructions doit être la même pour tous les contrôleurs JTAG des composants d'une même chaîne de scan. Les codes d'instructions non spécifiés sont librement définis par les concepteurs du système. En fonction des instructions chargées, différents registres de données peuvent être sollicités :

- **BYPASS** de taille 1, permet de court-circuiter le composant sélectionné dans la chaîne de scan, si l'on n'a pas besoin d'accéder à ses valeurs.

- **BOUNDARY-SCAN**, composé par les différentes cellules *Boundary-Scan* (voir section D.1.1), il est utilisé pour tester les continuités entre les composants électroniques.

- **IDCODE**, est optionnel, mais largement implémenté dans les contrôleurs JTAG. Il contient l'identifiant du composant.

Les registres de données restants permettent d'obtenir d'autres informations définies par les concepteurs du système.

D.1.6 Bus de débogage

En général, les fonctions de débogage ne sont pas implémentées dans le contrôleur TAP mais plutôt dans des IP-bloc dédiés. Par conséquent, les fonctionnalités de débogage sont spécifiques au système. L'accès aux IP-bloc se fait à travers le bus de débogage via des instructions JTAG. La communication est totalement prise en charge par des registres de données spécifiques et le contrôleur TAP sert uniquement d'interface.

11. Source : https://www.xjtag.com/about-jtag/jtag-a-technical-overview/

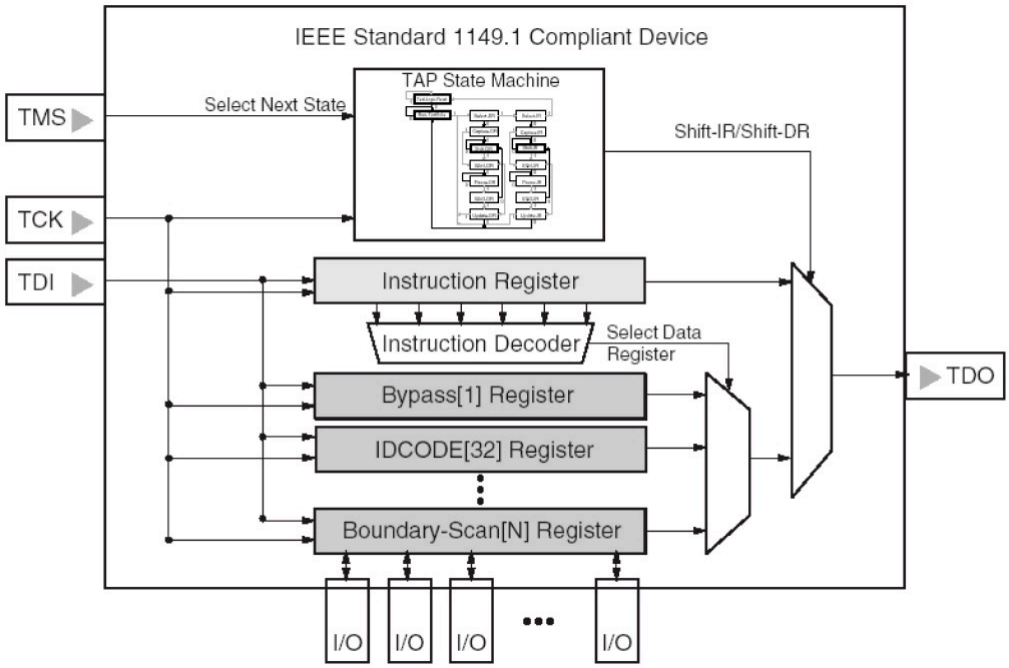


FIGURE D.5 – Architecture d'un contrôleur JTAG [11]

D.2 Le SWD

Le *Serial Wire Debug* (SWD) [138] est une solution alternative au JTAG proposée par ARM. Elle fait partie de l'ensemble d'IP spécialisés pour le débogage : ARM CoreSight® [41]. Il est possible d'accéder aux éléments de cette infrastructure à l'aide des outils de débogage externes au système via une interface utilisant seulement deux signaux. A l'instar du JTAG, le SWD permet la programmation et le débogage. Cependant le protocole n'implémente pas de solution pour réaliser du *Boundary-Scan* .

D.2.1 Les signaux

Le bus SWD est un bus série synchrone composé des deux signaux de contrôles suivants :

- **SWDIO**, un signal de donnée bidirectionnel, contrôlé à la fois par la sonde et le contôleur ciblé.

- **SWDCLK**, un signal d'horloge contrôlé par la sonde de débogage. Contrairement au JTAG, les données entrantes et sortantes sont toutes les deux chargées au front montant.

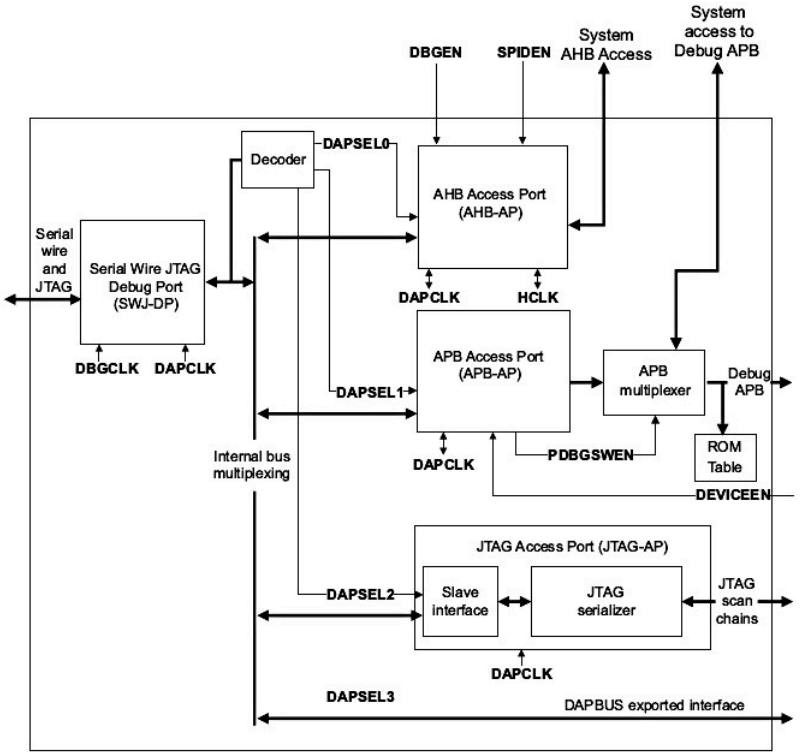


FIGURE D.6 – Architecture de débogage *Serial Wire/JTAG*. (Source :[41])

D.2.2 Architecture du SWD

Avec une architecture SWD, une sonde de débogage se connecte directement au contrôleur *Serial Wire Debug Port* (**SW-DP**), qui accède aux différents *Access Ports* (**SW-AP**) des modules du système. Pour proposer les mêmes fonctions que la norme IEEE 1149.1, les fabricants de processeurs intègrent les deux protocoles. Ainsi, comme l'illustre la figure D.6, un contrôleur *Serial Wire JTAG Debug Port* (**SWJ-DP**) fait office d'interface entre les outils de débogage externes et les protocoles de débogage du système. Un décodeur d'instruction aiguille les signaux en fonction du type d'opérations souhaitées.

D.2.3 Le protocole SWD

Le protocole SWD est lié à l'architecture ARM dans laquelle il est implémenté. Les trames échangées ont une taille fixée à 44 bits. Elles ne sont pas extensibles, ce qui donne l'avantage d'être simple d'utilisation et efficace en terme de cycles d'horloge à générer pour effectuer des transactions. La figure D.7 présente la structure d'une transaction SWD. Elle est composée d'une phase de requête envoyée par la sonde, d'un acquittement (**ACK**) renvoyé par la cible, et d'une phase de données.

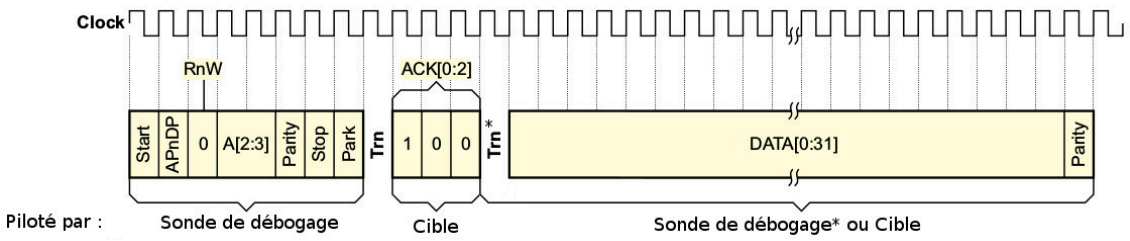


FIGURE D.7 – Transaction SWD (Adapté de [70])

SW-DP		
A[2:3]	Lecture	Ecriture
0x00	IDCODE	ABORT
0x04	CRTL/STAT	CTRL/STAT
0x08	RESEND	SELECT
0x0C	RDBUFF	N/A

AHB-AP		
A[2:3]	Lecture	Ecriture
0x00	CWS	CSW
0x04	TAR	TAR
0x08	N/A	N/A
0x0C	DWR	DWR
0xFC	IDR	N/A

Tableau D.1 – Exemple d'opérations sélectionnables en fonction de la valeur remplie dans le champ **A[**2:3**]** d'une requête. L'intégralité des valeurs et des opérations est détaillé dans [70]

D.2.3.1 La requête

La requête est définie par la sonde et renseigne les paramètres suivants :

- **APnDP**. Spécifie le port avec lequel on souhaite communiquer. L'Access Port (**AP**) pour accéder aux différents périphériques du système ; le Debug Port (**DP**) pour lire l'**IDCODE**, activer l'alimentation des ports de débogage et accéder aux différents **AP** (AHB, APB, etc.)

- **RnW**. Désigne s'il s'agit d'une lecture ou d'une écriture. Le bit de ce champ vaut "0" pour une écriture et "1" pour une lecture.

- **A[**2:3**]**. Spécifie le champ d'adresse du **DP** ou de l'**AP** que l'on souhaite atteindre pour effectuer un type d'action : écriture d'adresse, lecture mémoire, paramétrage, etc. Le Tableau D.1 donne des exemples de valeurs pour le **SW-DP** et le **AHB-AP**.

D.2.3.2 L'acquittement

L'acquittement **ACK[**0:2**]**, de taille 3 bits est commandé par le contrôleur ciblé. Il s'agit d'un statut sur l'état de la transaction :

- **OK (0b001)**. La transaction s'est bien déroulée.

- **Wait (0b010)**. Ce statut est renvoyé si l'on effectue une écriture à la suite d'une lecture sur le port **AP**. Ceci est dû au fait qu'une lecture sur le port **AP** s'effectue en deux temps. La première phase sert à charger les valeurs lues et la deuxième à les retourner sur le port **SWDIO**.

- **Fault (0b100)**. Une erreur a été détectée (ex. parité fausse). Pour effacer la détection il faut remettre à "0" le bit d'erreur situé dans le registre CTRL/STATUS en exécutant une commande ABORT.

D.2.3.3 Les données

Le champ des données est présent uniquement lorsque l'acquittement **ACK[**0:2**]** a la valeur **OK (0b001)**. Lors des écritures (**RnW** = 0), ce champ de 32 bits est commandé par la sonde de débogage. Lors des lectures (**RnW** = 1), il est commandé par le contrôleur ciblé. Dans les deux cas, un bit de parité est placé après le $32^{\text{ème}}$ bit de donnée.

Annexe E

Implémentations de l'AES

Cette annexe donne les pseudo-codes des implémentations de l'AES utilisées dans le chapitre 3.

Sommaire du l'annexe

E.1 Implémentation logicielle de l'AES

pseudo-code 3: AES LOGICIEL

Input: Clé de 16 octets : K, message de 16 octets : M, deux entiers : P_1 et P_2.
Output: Chiffré de 16 octets : C.

```
   /* Initialisation des variables */
1  C ← 0xCC ... CC;      //(16 octets de valeur 0xCC)
2  j ← 0;
3  GPIO("bas");

   /* Boucles "while" AVEC chiffrement AES */
4  GPIO("haut");      //voir section 2.3.1.2
5  while j < P2 do
6  |  i ← 0 ;
7  |  while i < P1 do
8  |  |  i ← i + 1;
9  |  end
10 |  C ← AES_ECB_128(K,M) ;
11 |  i ← 0;
12 |  while i < P1 do
13 |  |  i ← i + 1;
14 |  end
15 |  j ← j + 1;
16 end
   /* Boucles "while" SANS chiffrement AES */
17 while j < P2 do
18 |  i ← 0 ;
19 |  while i < P1 do
20 |  |  i ← i + 1;
21 |  end
   |  //rien
22 |  i ← 0;
23 |  while i < P1 do
24 |  |  i ← i + 1;
25 |  end
26 |  j ← j + 1;
27 end
28 GPIO("bas");
29 return C ; // Renvoie le chiffré sur l'UART de la carte de développement
```

E.2 Implémentation matérielle de l'AES

pseudo-code 4: AES MATÉRIEL DU CRYTPO-PROCESSEUR (C-HWA)

Input: Clé de 16 octets : K, message de 16 octets : M, deux entiers : P_1 et P_2.
Output: Chiffré de 16 octets : C.

```
   /* initialisation des Variables */
1  C ← 0xCC ... CC;      //(16 octets de valeur 0xCC)
2  j ← 0;
3  GPIO("bas");
   /* initialisation des registres du C-HWA */
4  C-HWA_Flag_Tâche_Prête ← False;
5  C-HWA_Flag_Status_Tâche_Faite = False; // En Lecture seulement
6  C-HWA_registre_Accelerateur_materiel_type ← AES-128-ECB;
7  C-HWA_registre_Accelerateur_materiel_mode ← Chiffrement;
8  C-HWA_registre_clé ← K;
9  C-HWA_registre_message ← M;

   /* Boucles "while" AVEC chiffrement AES */
10 GPIO("haut");
11 while j < P2 do
12 |  NOP;
   |  ...
111|  NOP;
112|  C-HWA_Flag_Tâche_Prête ← True; //Démarre le chiffrement
113|  while C-HWA_Flag_Status_Tâche_Faite ≠ True do
114|  |  //WAIT...//Le C-HWA retourne le chiffré dans 'buffer_C'
   |  |          //et monte le FLAG 'C-HWA_Flag_Status_Tâche_Faite'
   |  |          //à 'TRUE'
115|  end
116|  NOP;
   |  ...
215|  NOP;
216|  j ← j + 1;
217 end
   /* Boucles "while" SANS chiffrement AES */
218 while j < P2 do
219 |  NOP;
   |  ...
318|  NOP;
   |  //rien
319|  NOP;
   |  ...
418|  NOP;
419|  j ← j + 1;
420 end
421 GPIO("bas");
422 return buffer_C; // Renvoie le chiffré sur l'UART
                     // de la carte de développement
```

Annexe F

Listes des chiffrés <fauté>

Cette annexe donne la liste des fautes obtenues lors de perturbations EM injectées durant les chiffrement AES du chapitre 3. Ces listes sont issues des « résumés » de fichiers « `.log` » beaucoup plus volumineux que nous n'avons pas trouvé utile d'ajouter à ce document. Elles sont données afin de souligner la difficulté dans le tri et l'analyse des fautes. Les octets n'ayant pas été modifiés sont notés "`..`".

Sommaire du l'annexe

F.1 Fautes obtenues durant la perturbation EM de l'implémentation logicielle de l'AES

```
------------------------- REFERENCES --------------------------
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 (Message)
00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF ( Key )
AF E6 93 58 DE C5 71 93 28 2C 2F B6 B8 AB 62 16 (Cipher )
--------------------------- RESUME ----------------------------
0000: 46 93 D0 04 AA 9B D0 90 33 6B 94 32 64 54 .. D7 : 1 times
0001: EB FE 13 FC 85 46 70 17 C1 E1 3E D9 A4 CF 80 ED : 2 times
0002: 40 87 18 .. A3 80 .. 2C 1B .. .. .. .. 4E 8B 41 : 1 times
0003: 6A 0F 3F 14 E2 EE 77 2C 61 A3 F6 3C 6F CC 9B DB : 1 times
0004: 20 87 18 57 89 C2 9F F4 74 61 94 73 60 C9 20 C2 : 2 times
0005: 00 51 62 E4 E9 .. B9 34 BB C1 32 1C A9 C6 93 CB : 1 times
0006: 43 .. BF .. .. CB .. 78 4A .. 6B .. .. CC .. 41 : 1 times
0007: .. .. 50 3D .. C3 B3 .. 3B A5 .. .. EA .. .. B7 : 1 times
0008: 1B 02 AB 3A A2 02 76 46 89 9D A5 5E 83 CE 79 EB : 1 times
0009: 2A C2 .. .. E5 .. .. 0C .. .. F4 96 .. 8F C9 .. : 1 times
0010: .. C7 32 1C E6 C3 2F .. B0 79 .. BC EA .. 81 89 : 2 times
0011: 40 .. .. .. .. .. .. 2C .. .. .. .. .. 4E .. .. : 1 times
0012: .. .. 72 .. .. B7 .. .. 70 .. .. .. .. .. .. 8B : 1 times
0013: 4D 5F AB 5C A2 03 BA B0 BD A9 3E 32 22 EE C3 A3 : 1 times
0014: .. .. .. 1E .. .. 73 .. .. 75 .. .. A3 .. .. .. : 1 times
0015: 21 36 E9 A2 D0 EF 36 88 8A BD 23 73 2D 9B D4 84 : 2 times
0016: D4 EA E6 3C 7D D8 05 0C 48 4A 43 79 44 DC 05 A2 : 1 times
0017: .. 2F 5E 54 69 D1 73 .. 7A 9D .. 98 48 .. FD C3 : 1 times
0018: DA 86 63 7D E9 90 FB 28 BA E0 24 FF 6A CE 61 C2 : 1 times
0019: 06 C7 .. .. 65 .. .. 26 .. .. B6 3F .. 8F D2 .. : 1 times
0020: 00 5F BB 50 A2 E1 5D 37 71 29 B7 32 6C CA C3 0B : 1 times
0021: CB C5 6F 4A .. 89 04 84 BA 75 DB 2C 2C 4E FF C3 : 2 times
0022: 0A .. 72 13 .. E3 31 .. 3B 6D 8A .. 6F 8E .. 85 : 1 times
0023: 8A 45 .. .. A9 .. .. 2C .. .. E4 76 .. CC 81 .. : 1 times
0024: C3 CB 72 D4 9D 23 9C A4 1A 2B F5 B4 2B 0D 40 A7 : 1 times
0025: 08 8F 92 B4 CB DB B7 .. 92 68 3E 36 6F 69 48 CB : 1 times
0026: 42 .. 60 .. .. 83 .. .. FA .. A6 .. .. C6 .. C3 : 1 times
0027: CE .. .. .. .. .. .. 78 .. .. 78 .. .. 9E .. .. : 3 times
0028: AA .. .. 7D .. .. F3 7E .. 79 04 .. E2 CC .. .. : 1 times
0029: 77 70 A1 54 1F 32 B4 54 C4 07 46 21 24 77 15 0A : 2 times
0030: A0 98 BE F6 A3 0C 62 20 9A 4D A2 F5 00 45 81 29 : 2 times
0031: C9 C9 91 B4 E2 4D 26 E2 51 23 03 66 48 06 A9 57 : 1 times
0032: 07 0E .. .. 2D .. .. AC .. .. AC A6 .. 21 C1 .. : 1 times
0033: 0A .. .. 56 .. .. 32 D4 .. 68 AC .. 2B CE .. .. : 2 times
0034: 12 .. .. D1 .. .. B9 28 .. 69 F2 .. 2E 8C .. .. : 1 times
0035: 22 .. .. 1E .. .. 73 2C .. 75 6E .. A3 EC .. .. : 1 times
0036: 48 .. 72 .. .. 4A .. 38 3E .. 6E .. .. 44 .. 02 : 1 times
```

```
0037: EB D7 48 50 68 C1 49 01 2A 49 17 B6 3B AC 5E D1 : 1 times
0038: 6C .. .. 1E .. .. 72 6C .. 78 04 .. 63 8D .. .. : 1 times
0039: 4B .. 30 .. .. 81 .. 68 3A .. E5 .. .. 64 .. 8B : 1 times
0040: 1B AE B4 58 3A 20 FF 2C C9 24 BE 3C F0 54 A6 A0 : 1 times
0041: .. 0F 91 72 68 E3 7A .. 72 69 .. 1D 57 .. A9 66 : 1 times
0042: 35 8D 44 41 C3 D1 CE 2D 92 6D E0 EB 6F 21 3D E2 : 1 times
0043: 0E C7 12 10 EC CB 98 2C 1A 49 2F .. 23 CC 91 C7 : 2 times
0044: .. .. .. F1 .. .. F1 .. .. 29 .. .. 68 .. .. .. : 3 times
0045: 06 8F .. .. 28 .. .. 26 .. .. B6 3A .. 8F C9 .. : 1 times
0046: 51 8B 1E ED B6 43 B3 F1 F3 87 BC FF 04 EC A3 D3 : 1 times
0047: F5 A4 F7 9C C5 41 1C 6E 28 C8 8A 70 DF CC 82 90 : 1 times
0048: 72 CF 52 .. E9 C1 .. 66 36 .. A3 B7 .. 25 85 9B : 1 times
0049: 1A 19 7E 1A 01 C3 B7 2E DA 2A EE BF 2B CC 41 D3 : 1 times
0050: 41 C2 70 1C E5 49 13 34 27 07 A7 96 24 E4 C9 A3 : 1 times
0051: 2D C7 CE 3D FC 4A 56 75 B7 6B F2 F8 A8 A6 7D 42 : 1 times
0052: 07 .. 73 .. .. 03 .. AC B0 .. AC .. .. 21 .. D7 : 1 times
0053: .. 61 73 .. A0 03 B3 .. B0 B9 .. .. EA .. E2 D7 : 1 times
0054: 26 .. 76 .. .. E3 .. 1C 30 .. A6 .. .. C8 .. C1 : 1 times
0055: AE 0A .. BF E1 DB EF C9 88 6D 32 19 4A 8E A0 76 : 1 times
0056: A5 72 .. 09 E4 7B 76 5B D3 C9 84 D0 70 0D 85 13 : 1 times
0057: B7 FA 9A 52 BA 5F 7A 24 18 E3 AA 5F 68 C6 91 A4 : 1 times
0058: D3 81 1B 9A A0 DF 3B 0D 9B 81 EF 7E F0 C8 E5 D2 : 1 times
0059: 40 8E 6C 48 96 C3 31 AE 2F 18 A3 7A F6 C0 B1 0B : 1 times
0060: 40 2E 20 9D 87 .. BB 2C FA 49 .. D3 3B 4E E0 BF : 2 times
0061: 88 07 BA 9A E8 0B 50 24 F8 39 A5 3F E2 04 95 D1 : 1 times
0062: 46 C5 AA .. ED EF .. 68 FB .. F2 B2 .. D4 89 97 : 1 times
0063: AD 67 1A 55 F8 CF 57 2C 7D 4D 4B 79 BE CC 6E 81 : 1 times
0064: 0A .. 13 .. .. C3 .. AE 2E .. 86 .. .. DD .. 67 : 1 times
0065: 4D 5F 07 7F A2 C3 1F E4 AE C9 06 0B BD CC 6F E3 : 1 times
0066: 28 73 5A A7 C7 63 94 E4 A8 87 1B 1B 4B C2 BB 4E : 1 times
0067: .. .. .. 14 .. .. D1 .. .. 49 .. .. A7 .. .. .. : 2 times
0068: 46 .. BA .. .. C3 .. AE B2 .. FD .. .. AE .. C7 : 2 times
0069: C3 E4 16 4A B0 C3 3E 4F 71 49 2E 5E 2A CC A0 89 : 1 times
0070: D4 52 14 5C B6 E3 56 6F 26 6E A6 3F 74 8C C2 C3 : 1 times
0071: 09 60 BA 7A A2 C2 F3 EC .. 4F B2 BC 04 07 8B C3 : 1 times
0072: 20 .. .. 1E .. .. 73 34 .. 75 C6 .. A3 64 .. .. : 1 times
0073: 4A 86 26 0C E9 4A F2 3E 3B 6C AF FF 23 C8 61 E3 : 1 times
0074: .. 86 E0 .. C8 .. .. .. 5E .. .. 9D .. .. 93 6B : 1 times
0075: 02 .. .. .. .. .. .. .. 88 .. .. A6 .. .. C0 .. .. : 6 times
0076: 4D .. 9A 5C B2 87 BA E4 59 A9 AA 5A 22 CC 93 A3 : 1 times
0077: CA CF 52 .. 28 C7 .. E4 7B .. 8E BC .. A6 A1 C1 : 1 times
0078: 45 5F BA 5C A2 0B BA A4 F8 A9 2F 32 22 EC C3 D1 : 1 times
0079: .. 0E .. .. F2 .. .. .. .. .. .. 1E .. .. CD .. : 4 times
0080: 22 0D FA 0B D9 4B 73 AF E2 A1 0E 1F EA 06 95 8B : 2 times
0081: 49 6F 75 7B B2 2F 70 3D D2 07 BC BC 41 41 A1 E2 : 1 times
```

```
0082: 0A 8D 60 1C E9 83 2F .. FA 79 EA BC EA D4 49 C3 : 1 times
0083: 28 DB 88 D0 97 C3 92 65 31 E1 26 A6 E8 98 0D 83 : 1 times
0084: .. CE 50 82 68 51 FF 0E 96 79 E4 B0 27 .. 55 81 : 1 times
0085: 80 .. 9B 98 B2 C5 67 25 33 E9 A0 38 41 AE 81 41 : 1 times
0086: 44 .. 13 D4 .. D7 93 6E 12 6D 2F .. E9 68 .. 0B : 1 times
0087: .. D5 12 .. EB E0 .. .. F2 .. .. 31 .. .. D5 A2 : 1 times
0088: .. 16 .. .. 41 .. .. .. .. .. .. .. .. .. A1 .. : 2 times
0089: DA 1F 63 1C 88 90 DB 28 BA 6A 24 72 2A CE .. C2 : 1 times
0090: 62 45 7E DE DB 87 9A 88 1F E9 FA .. BE CA CC EB : 3 times
0091: A3 86 5A 54 C8 0F A9 24 5B 20 8A 9D A3 8E 93 83 : 2 times
0092: 4A AB 26 25 AD AB BA BD 12 00 FA 3C A3 84 85 8A : 1 times
0093: DC C3 71 72 .. C3 70 AD B3 61 EE 3E 2F C5 83 82 : 1 times
0094: .. CB 72 D0 AD 0E 73 .. 3A 29 .. 5F 1B .. A9 01 : 1 times
0095: 0A C5 9B B4 AB 57 B7 .. 3A 68 8A 7C 6F 8E E1 E4 : 1 times
0096: 48 D2 D6 5E 2D 83 70 2C 3B 49 2E CA 29 C3 D0 E0 : 1 times
0097: C3 C2 AA 1C 95 .. 7A A4 3E 19 F5 25 A2 0D 4C 97 : 2 times
0098: 82 07 05 09 B6 86 38 D6 31 0B 3F FF 04 2C 09 07 : 4 times
0099: 06 .. .. .. .. .. .. 26 .. .. B6 .. .. 8F .. .. : 2 times
0100: 4D 48 C3 73 E9 85 F6 81 72 A9 E6 59 64 CC CD E8 : 2 times
0101: 4D 5F BB 54 A2 DB 02 E4 56 29 AA 32 89 CC C3 A7 : 1 times
0102: 0D .. .. 08 .. .. B2 84 .. 60 24 .. 2F 0D .. .. : 1 times
0103: .. CB .. C4 E0 .. 33 .. .. 4A .. 3C 8A .. 85 .. : 1 times
0104: .. E5 B0 .. 7F 02 .. .. 50 .. .. 23 .. .. A3 C7 : 1 times
0105: 38 07 3C 9F E6 30 33 2E 37 9A C2 D3 1D 97 B5 16 : 1 times
0106: AB 30 69 76 E2 A0 4E DC 15 4F C4 EA 42 29 43 6B : 1 times
0107: CE C5 F0 3D ED 91 18 78 B2 6D 78 B2 08 9E 89 C3 : 1 times
0108: 4D 5F 77 5C A2 8A 5C 2C DF 4F CE 3F 0E CC A7 E4 : 1 times
0109: 03 6F F7 D5 8A BF 7A A1 5C C0 44 BC AD 7A 68 90 : 1 times
0110: 4B C7 DA 1C F1 .. 2F 58 32 79 24 BC EA 64 48 C7 : 2 times
0111: 12 5F 32 72 E9 CC 70 58 70 61 A2 34 2F E4 15 4B : 1 times
0112: 0A .. .. .. .. .. .. .. .. .. 8A .. .. 8E .. .. : 1 times
0113: 40 45 .. .. 61 .. .. 2C .. .. .. 3A .. 4E 00 .. : 1 times
0114: 49 52 AB 43 01 EE 67 2E 71 A5 45 3E 4C 64 50 C9 : 1 times
0115: 1E 45 3A 5E 01 87 6A FD B2 6D 2E 1E 33 64 85 C3 : 1 times
0116: 6D 52 BA 5C 01 0B BA 20 F8 A9 A6 3E 22 CC 50 D1 : 1 times
0117: 4A 5F BA 5C A2 0B BA 64 F8 A9 A3 32 22 21 C3 D1 : 1 times
0118: 68 6C 42 8B D5 74 4A 99 4C 54 BD 4B 6A DB CA 20 : 1 times
0119: 4D 60 9B 7A A2 D8 F3 AC 53 4F BB BC 04 AC 8B C5 : 2 times
0120: 29 64 81 14 82 C2 C6 23 8F 1A 2F 38 50 8E 15 2A : 1 times
0121: .. 86 55 .. C8 E3 .. .. A6 .. .. 9D .. .. 93 D7 : 1 times
0122: 42 C7 9A 68 60 66 DB 28 72 6B AC 2A E9 C8 C9 C2 : 1 times
0123: 2E .. A5 59 04 B7 26 A6 3B 6B AA EA A2 9D D7 4A : 1 times
0124: 06 .. 72 .. .. 41 .. 68 D9 .. A6 .. .. 68 .. BF : 1 times
0125: D5 5C 77 D1 A0 41 55 2E 30 E9 A6 3E E2 0E CB CB : 1 times
0126: .. 85 DA 9D A1 49 D1 .. B6 A8 .. 50 E9 .. C9 C2 : 1 times
```

```
0127: 80 0A EA 13 F9 03 56 E8 3E CD 0E 23 03 CC 4C AD : 1 times
0128: .. 85 .. D8 EB .. AA .. .. E1 .. 11 EB .. 81 .. : 1 times
0129: E5 96 AB 07 2A 14 70 6B F9 E8 .. 64 22 DC 68 44 : 2 times
0130: 4B 55 BA 9A E0 0B 50 38 F8 39 BC 26 E2 DC 91 D1 : 1 times
0131: 06 C7 7A D0 C9 02 DA 6E 1B 6D A6 2E 6F D1 81 5B : 1 times
0132: 00 .. .. 95 .. .. BF 34 .. 6B 32 .. 03 C6 .. .. : 1 times
0133: 5A 64 B2 5E 63 07 BD 77 28 71 2E .. BF 84 49 87 : 1 times
0134: 07 61 73 3C A0 03 3B AC B0 20 AC .. E9 21 E2 D7 : 1 times
0135: .. 6F .. 13 61 .. 31 .. .. 6D .. 1F 6F .. A5 .. : 2 times
0136: 4D 43 BA 5C 4B 0B BA E4 F8 A9 AA BC 22 CC 85 D1 : 1 times
0137: 06 D3 72 D4 6B B7 9E 26 70 72 B6 96 6A 8F 9D 8B : 1 times
0138: 0A A7 73 7F A9 .. F3 0C 3B AD A0 2E AA C0 C1 43 : 1 times
0139: .. A3 30 .. C1 81 .. .. 3A .. .. 7E .. .. E1 8B : 1 times
0140: 90 C6 .. .. E9 .. .. 0C .. .. 02 3C .. C3 C5 .. : 1 times
0141: C1 5F F1 5C A2 41 BA 8A B9 A9 A2 32 22 CC C3 57 : 1 times
0142: 0E C6 7A 1C E9 02 13 68 1B 07 8E 3C 24 00 C5 5B : 1 times
0143: 07 .. 70 .. .. 57 .. AC 2E .. AC .. .. 21 .. C1 : 1 times
0144: 16 .. .. 90 .. .. AF B4 .. 79 66 .. 2B 54 .. .. : 2 times
0145: 0A C7 3A 08 C9 49 AF .. 36 69 8A 2E 6A 8E 81 C7 : 1 times
0146: D4 0E 59 5C A4 4A C2 6F 31 68 E6 5E 16 DC A0 8B : 1 times
0147: .. C6 .. C2 E8 .. 43 .. .. 72 .. 4E 22 .. A3 .. : 1 times
0148: .. 87 72 .. E0 B7 .. .. 70 .. .. FF .. .. 81 8B : 1 times
0149: 2D 07 7F BF 82 D8 9C 0A .. AD 72 58 E2 A6 7D D0 : 1 times
0150: 42 A1 E9 5C AC DF 30 2C 32 1E AA E6 BA D8 C9 C3 : 1 times
0151: F2 C3 E4 4D 18 91 3B 8E 18 79 A0 E6 EA 05 D1 D3 : 1 times
0152: EC 44 BC 2B 85 29 31 AA 71 0B 21 2C .. CA C2 D3 : 1 times
0153: C3 C4 91 08 E9 E3 AF 0E 72 69 A0 2C 6A E4 48 66 : 1 times
0154: CB 5F BA 1C A2 0B 38 30 F8 21 A3 32 78 0D C3 D1 : 1 times
0155: .. 61 .. .. A0 .. .. .. .. .. .. .. .. .. E2 .. : 1 times
0156: C4 3E AB FD AA FD 32 C3 99 CB 81 3C 45 8C .. 6E : 1 times
0157: 56 C5 7E 3E 41 C0 DF 28 92 2B AC 3A 08 6E E1 C3 : 4 times
0158: 03 C4 9B 13 E9 57 56 58 3A CD E4 2C 03 CD 48 E4 : 1 times
0159: 01 E8 6B D8 A2 8B BE 58 38 0D 6E .. 21 CC E2 C7 : 1 times
0160: .. 85 79 .. AF C1 .. .. 58 .. .. 6D .. .. 27 01 : 2 times
0161: .. .. .. 1C .. .. DB .. .. 6A .. .. 2A .. .. .. : 1 times
0162: EF BE D0 59 82 89 65 ED .. 37 26 7A 40 C0 B1 C3 : 1 times
0163: .. .. .. 13 .. .. 31 .. .. 6D .. .. 6F .. .. .. : 5 times
0164: 06 C3 .. .. EB .. .. 26 .. .. B6 7C .. 8F 99 .. : 1 times
0165: 07 CB .. .. EC .. .. .. .. .. D2 22 .. D8 80 .. : 1 times
0166: 28 85 06 08 0D .. D7 50 3A 2B C3 FB 6F CD E5 EB : 1 times
0167: 10 .. .. .. .. .. .. 58 .. .. 05 .. .. 4C .. .. : 1 times
0168: 08 C3 F1 7B E0 0F 72 45 B8 E6 F7 5F A8 EE 27 A0 : 1 times
0169: C5 A0 B9 1B A7 C3 29 D5 7D 4D AE 7D E8 88 81 F6 : 3 times
0170: 2D 67 9B 7A 81 D8 F3 8E 53 4F 96 36 04 A6 81 C5 : 1 times
0171: 16 .. .. .. .. .. .. 66 .. .. EE .. .. CA .. .. : 1 times
```

```
0172: E0 66 6C 75 2C C2 40 A2 8F 40 F9 85 48 6D C6 A8 : 1 times
0173: 27 36 .. D8 E9 17 3A 2F 71 E1 25 3E E9 90 88 .. : 1 times
0174: 48 C7 73 0E A1 97 AF F2 FE 68 A6 BC 24 CC 20 CB : 1 times
0175: 02 B3 57 1F A7 65 BE 2C 15 09 25 6E 68 CC A8 C2 : 1 times
0176: 4B F7 BA 56 F9 C3 D9 58 B2 46 24 36 .. 64 CB C7 : 2 times
0177: CB .. AA .. .. .. .. 58 3E .. AC .. .. ED .. 97 : 2 times
0178: .. CF 92 .. 28 23 .. .. B2 .. .. BC .. .. A1 8B : 1 times
0179: 8A .. BA .. .. C3 .. 38 B2 .. 2C .. .. CC .. C7 : 1 times
0180: 02 C5 AA BF 87 .. 93 6E 3E E1 A2 1D 6F A2 89 97 : 1 times
0181: .. .. .. 90 .. .. AF .. .. 79 .. .. 2B .. .. .. : 5 times
0182: 12 85 5E 0C EB 6B F2 4C 79 6C 2E 11 23 C4 81 DF : 1 times
0183: 07 .. 73 68 .. A1 DB AC FA 6B AC .. E9 21 .. D7 : 1 times
0184: 7D 94 2B B0 2E EB 2B 6D 3E 62 E2 3C AC CC 91 CB : 1 times
0185: .. 0E F0 1E 2D 5B 43 .. 36 23 .. A6 6F .. C1 C2 : 1 times
0186: C0 5F BA 5C A2 0B BA FC F8 A9 B6 32 22 CD C3 D1 : 1 times
0187: 07 .. BA .. .. C3 .. AC B2 .. AC .. .. 21 .. C7 : 1 times
0188: 75 CE E2 74 9A C2 9B CF 2A EC AF F7 E9 CC 93 2A : 1 times
0189: E2 C6 .. .. EC .. .. B4 .. .. D6 F3 .. AF 01 .. : 2 times
0190: 17 41 9A 5B FC 8B 9F 7E 71 A9 B3 DC 22 CD A8 82 : 2 times
0191: 48 57 BA 5C E2 0B BA 0E F8 A9 .. 7E 22 .. C2 D1 : 1 times
0192: 4C 5F 77 5C A2 41 BA A4 30 A9 86 32 22 8C C3 CB : 3 times
0193: C9 60 BB 32 A2 C7 7A F2 D9 A8 26 BC 60 A6 8B 6B : 1 times
0194: 49 BC 76 73 F1 14 7A 63 38 49 E5 1B 2E CE 93 EC : 1 times
0195: 82 4F 72 .. 48 80 EF 2E 3C 6D A7 3E 21 CD C5 C9 : 1 times
0196: 68 C6 FD 58 11 CA BD E5 D1 29 C2 5E D0 D4 A0 C1 : 1 times
0197: 81 B2 44 10 EB A2 E0 24 7A FE A0 3F 09 44 08 C3 : 1 times
0198: .. 07 .. .. E9 .. .. .. .. .. .. EE .. .. 83 .. : 1 times
0199: 89 5F AB 5C A2 E3 BA 4C FB A9 3E 32 22 D9 C3 89 : 1 times
0200: 0A 61 .. .. A0 .. .. .. .. .. 8A .. .. 8E E2 .. : 3 times
0201: 1E 45 9B .. 6B 57 B3 .. 3A B9 24 72 EA 9D AC E4 : 2 times
0202: 6D .. BB 5A 6A DB 5D 66 56 FB A0 B6 A2 58 81 A7 : 1 times
0203: 4D 5F 9A 10 A2 87 50 E4 59 C9 AA 32 22 CC C3 A3 : 1 times
0204: 4D 04 E9 10 2A D1 7C E4 F8 EB 96 3C 41 CC A6 C2 : 1 times
0205: 49 4B BB 7A C8 E0 F3 4C 6A 4F AF 2E 04 EE 81 A7 : 1 times
0206: 0B BB 71 56 A2 79 9F 2C 70 81 24 3E E8 25 D3 87 : 2 times
0207: 48 CF F0 .. E9 91 .. F4 B2 .. AE F6 .. A2 80 C3 : 1 times
0208: 07 .. .. .. .. .. .. AC .. .. AC .. .. 21 .. .. : 1 times
0209: 2D 60 69 D9 A2 D8 A0 AE 89 69 64 36 FD A6 79 C5 : 1 times
0210: 2D 60 F1 93 A2 41 77 8E 31 A1 96 BC A1 A6 8B C3 : 1 times
0211: 40 CB BF 1E 9D C0 43 C5 D9 23 AC B4 6F DE 40 F3 : 1 times
0212: 06 F3 71 1D E1 89 D5 26 FB 71 B6 3E 2B 8F A3 8A : 1 times
0213: EB 08 65 3E 07 DC 32 89 F9 A9 C7 DA E8 4C C5 7E : 1 times
0214: .. C5 .. D1 ED .. B9 .. .. 69 .. B2 2E .. 89 .. : 1 times
0215: 06 33 BF 9C FD C6 DB 36 91 87 2A 31 69 AC 05 03 : 1 times
0216: 4B 5F BA 5C A2 0B BA 4C F8 A9 2E 32 22 CC C3 D1 : 1 times
```

```
0217: 4B 29 1A 7B A0 6D 94 38 7A 15 1B CC 59 C2 38 92 : 1 times
0218: 4D 83 9B 7A E6 D8 F3 AC 53 4F BB 36 04 AC 0B C5 : 1 times
0219: 46 .. .. 41 00 0F FA C2 51 29 B8 BC 64 45 C5 01 : 1 times
0220: .. C3 .. .. AD .. .. .. .. .. .. 3C .. .. EF .. : 1 times
0221: A2 C2 42 D0 A5 .. 93 AE 7E B9 22 72 EA CC A8 97 : 1 times
0222: 94 .. 12 .. .. C3 .. 50 70 .. A6 .. .. AD .. C3 : 1 times
0223: 25 .. .. .. .. .. .. 20 .. .. 6E .. .. 1C .. .. : 1 times
0224: 1A .. .. .. .. .. .. E4 .. .. A6 .. .. 0D .. .. : 1 times
0225: 07 4E EA 9D E9 91 BB AC 32 49 AC 2E 3B 21 C9 80 : 1 times
0226: .. .. 11 1C E9 C3 DB .. 17 6A .. 7D 2A .. A2 C1 : 1 times
0227: 1A .. .. 3D .. .. B3 18 .. A5 F7 .. EA C8 .. .. : 1 times
0228: 56 84 94 00 A7 0B 97 B4 17 20 0E 3E E9 C9 26 C1 : 1 times
0229: 9C C4 D1 5A FB B3 58 0B 2C 6A A6 DD 31 CC 45 E3 : 1 times
0230: 64 23 D9 5C 0E .. BA 6F 5C 00 E4 2A 86 CE 96 D1 : 1 times
0231: AC CE .. 30 A6 09 39 4E 3D 68 60 1F ED 24 A1 .. : 1 times
0232: 08 4E A3 5C E3 C2 83 0B 71 8D 0E CA 70 6D C9 C3 : 1 times
0233: 12 .. 10 88 .. 81 BF 26 3A 6D EA .. 2F D4 .. E3 : 1 times
0234: C2 .. D2 69 21 89 7A FC 89 E9 CC 1D E8 6A F5 43 : 1 times
0235: EA 5F BA 12 A2 0B 76 2C F8 BD .. 32 6C CC C3 D1 : 1 times
0236: 46 83 62 B4 E8 CB B7 AE A6 68 FD 1F 6F AE 80 97 : 1 times
0237: BD C7 11 37 A7 26 72 A0 61 26 0E 77 A2 AE 81 4B : 1 times
0238: E1 44 23 54 68 82 32 4C 71 28 C7 A2 62 4E 79 C0 : 1 times
0239: 08 .. D1 BD .. C3 F3 D8 3B 5D EA .. A9 8F .. DF : 1 times
0240: B1 83 C0 FA F1 C3 27 2E 3A 68 36 79 6F CC 15 AD : 1 times
0241: 94 D5 .. .. 6B .. .. 50 .. .. A6 3E .. AD 81 .. : 1 times
0242: 99 73 63 C7 7A C7 8A D4 63 AD 4C 8D F4 58 7D A1 : 1 times
0243: 49 5F BA 99 A2 0B BB 30 F8 99 85 32 60 AD C3 D1 : 1 times
0244: 66 05 3A 13 E3 49 31 88 36 6D 78 3A 6F 44 09 C7 : 1 times
0245: 8A 4F 18 DD ED 80 1D 58 1B 3B E2 A8 39 A8 09 41 : 1 times
0246: 43 CF AB 6D 1D 87 32 5D 9E 2D D2 3D 27 70 79 81 : 1 times
0247: 22 .. .. B4 .. .. F8 AC .. 4B 86 .. A9 A8 .. .. : 1 times
0248: 7B 45 9A .. B2 C2 F3 EC 8F 4E 96 1A 4F A7 99 A1 : 1 times
0249: .. 87 8A .. E0 A0 .. .. B2 .. .. FF .. .. 81 AD : 1 times
0250: E2 0A 29 3C ED 57 3B B4 B2 20 D6 DD E9 AF 45 13 : 2 times
0251: 18 52 BA 5C 01 0B BA 66 F8 A9 E4 3E 22 EB 50 D1 : 1 times
0252: 02 03 70 BD 4A .. C7 6E 1B 21 A2 9D 2E A8 C1 D3 : 1 times
0253: 07 C7 13 7F C9 C3 F3 AC 2E AD AC 2E AA 21 81 67 : 1 times
0254: CE 62 72 14 E9 4A BA 28 3E F5 83 36 57 C6 09 02 : 1 times
0255: 5A .. 31 .. .. C2 .. 77 3E .. 2E .. .. 84 .. CA : 1 times
0256: 08 8B F3 1E 1D 0F 53 03 3A 6D 86 3D 2F 04 8B C7 : 1 times
0257: F3 5F 1B 50 D9 1D DF E8 3C 19 43 F2 A2 C8 06 D7 : 1 times
0258: 4D 5F BF 5C A2 C9 BA E4 32 A9 AA 32 22 CC C3 C7 : 1 times
0259: 4D 55 54 EC 3C 8A 35 25 BD 22 ED 89 72 CC C6 F0 : 1 times
0260: 7E .. .. .. .. .. .. 48 .. .. A6 .. .. CC .. .. : 3 times
0261: C3 86 3A 21 C2 52 12 0F 0C 6A CC 8A 1A E7 5C 66 : 1 times
```

```
0262: .. 07 .. 14 E9 .. FF .. .. 72 .. EE 83 .. 83 .. : 1 times
0263: .. 45 FA .. A9 B3 .. .. 1A .. .. 76 .. .. 81 E1 : 1 times
0264: 4D .. 9A 5A B2 2B 5D 24 71 FB A6 5A A2 C4 93 83 : 1 times
0265: 20 83 5E .. E8 D1 .. 04 7A .. A7 1F .. C8 80 C3 : 2 times
0266: 0A EF 50 7D E1 C3 F3 A4 3B 79 .. 1D E2 DE 13 B7 : 1 times
0267: 42 0D 55 DC A9 9B B9 A0 .. 6B 2E 76 EA 80 49 83 : 1 times
0268: 78 0D F3 2F E4 CD 23 59 18 E3 A7 5E C8 CC A3 B2 : 1 times
0269: 49 C7 A9 40 F0 6B 0E 2D BD A9 E6 1D 28 68 75 DB : 1 times
0270: 77 37 53 14 2C 8A 06 6E 77 EA A6 FE 45 ED 43 41 : 1 times
0271: B1 55 29 16 B0 C7 A6 84 30 C6 37 B8 72 4E 01 1D : 1 times
0272: 25 EF BE 1D E9 41 F9 2F 4E 69 .. 26 39 CE 9D 12 : 1 times
0273: .. C3 .. .. EB .. .. .. .. .. .. 7C .. .. 99 .. : 1 times
0274: C9 .. 32 9D .. C3 BB 34 B0 49 BC .. 3B EB .. 89 : 1 times
0275: 0E .. .. 13 .. .. 31 68 .. 6D 8E .. 6F 00 .. .. : 1 times
0276: 20 87 F3 0E E0 0F 3A 2D 3A A5 B7 FF 28 6F 81 C7 : 1 times
0277: 93 6F EE 12 D0 4A C6 E5 98 CD 26 17 03 8F 35 DB : 1 times
0278: 46 D6 9A 3E A1 66 FA AE 72 65 FD 7E 0A AE 89 C2 : 1 times
0279: C3 .. 62 .. .. CB .. 0E A6 .. A0 .. .. E4 .. 97 : 1 times
0280: 03 C3 12 D8 AD CB FA E4 1A 71 B6 3C 37 4E EF C7 : 1 times
0281: 6C CF A2 1C E1 C7 BA 28 72 7D AC 3F 23 8E 85 CB : 1 times
0282: .. C4 91 1C E9 E3 7A .. 72 19 .. 2C A2 .. 48 66 : 2 times
0283: AA 66 72 94 28 E1 73 7E E2 6D 04 77 22 CC 8A C2 : 1 times
0284: 48 4E 61 42 AA 7A 72 2E 30 E0 56 32 E0 CB 08 C3 : 1 times
0285: 7E C5 12 13 AB C3 56 48 70 CD A6 7C 03 CC E1 C3 : 2 times
0286: 4D 5F EA 5C A2 B3 BA 4C 20 A9 32 32 22 8C C3 0B : 1 times
0287: 48 .. .. 1C .. .. D1 2C .. 2B 2E .. 6A C3 .. .. : 1 times
0288: 4D 5F E2 74 A2 C2 9B C5 12 EC A0 A2 D1 CC C9 2A : 1 times
0289: 7E 27 D1 BF ED C3 93 48 36 E1 A6 2A 6F CC 83 E3 : 2 times
0290: CE .. .. 56 .. .. BB 78 .. 07 78 .. 29 9E .. .. : 3 times
0291: 06 D3 9F BD E9 83 F3 39 35 5D BE 23 A9 8E 95 90 : 1 times
0292: 4C 5F BB 5C A2 E1 BA A4 71 A9 86 32 22 8C C3 0B : 1 times
0293: 41 91 77 5C 2A 67 BA 24 70 6B 0D 3F AA C6 2B D1 : 1 times
0294: .. E5 .. .. ED .. .. .. .. .. .. .. .. 09 .. : 1 times
0295: 7B 79 BE F2 E3 71 7E 66 5C 69 BE D9 60 D7 C9 21 : 1 times
0296: 07 CF .. .. 21 .. .. AC .. .. AC 34 .. 21 C3 .. : 1 times
0297: 48 .. .. .. .. .. .. 2C .. .. 2E .. .. C3 .. .. : 2 times
0298: DA 9C 32 2C E9 C2 FB 4C 3F 9D F4 5E AB 21 88 01 : 1 times
0299: 08 2E B0 C4 2B 02 33 03 50 4A 86 3C 8A 04 8E C7 : 1 times
0300: 4D 5F BB 5C A2 83 BA E4 61 A9 AA 32 22 CC C3 C1 : 2 times
0301: .. C3 .. .. E9 .. .. .. .. .. .. 3C .. .. 40 .. : 1 times
0302: 12 87 71 16 E8 C5 BB 2C B8 72 6E 3C 63 0E A1 EC : 1 times
0303: 10 87 .. .. E0 .. .. B8 .. .. 24 FF .. 38 81 .. : 1 times
0304: F8 1F 2B C1 16 5F E6 2C 15 C4 37 F0 00 04 13 E3 : 1 times
0305: 1A .. 7A D4 E9 4B BA 28 22 68 04 7D .. CE A2 02 : 2 times
0306: .. 87 72 56 A3 4A EF .. B3 5D .. .. 27 .. 8B C3 : 1 times
```

```
0307: .. D5 72 .. EB E2 .. .. 32 .. .. 31 .. .. D5 8F : 1 times
0308: .. C6 .. .. E8 .. .. .. .. .. .. 4E .. .. A3 .. : 1 times
0309: 4D F1 AE D2 2A E7 70 E7 2E A3 9A 97 60 CD 48 A2 : 3 times
0310: 07 F3 30 9D E9 C3 D1 AC 3A A8 AC 3E E9 21 A9 F3 : 1 times
0311: 69 2B 37 90 A2 6B 80 EF 5D B3 CF F1 34 C6 C7 F7 : 1 times
0312: 09 .. B3 2C 6A 0F 6A B0 33 F5 A3 B6 21 44 81 13 : 1 times
0313: .. 51 .. 90 C4 .. AF .. .. 79 .. BE 2B .. A0 .. : 1 times
0314: 51 C3 72 .. B2 D7 BB 28 28 6D A7 9C 27 DC 62 C3 : 1 times
0315: 2A C5 BF 7E EF C0 A6 0C D9 A0 F4 A2 21 8F C9 F3 : 1 times
0316: .. .. 5E .. .. D1 .. .. 7A .. .. .. .. .. .. C3 : 5 times
0317: CB .. .. 13 .. .. 31 58 .. 6D AC .. 6F ED .. .. : 1 times
0318: 07 .. .. 1C .. .. 2F AC .. 79 AC .. EA 21 .. .. : 1 times
0319: 22 83 34 F1 E5 C3 BF AC 3A CD 86 DD 2B A8 40 E2 : 1 times
0320: 82 0C FF EC 25 5A AB 61 BA A3 04 3A 0F 8C A0 8F : 1 times
0321: 56 C2 70 1D E5 57 FA 2C 2E 66 FD 96 2A CC C9 C1 : 1 times
0322: 02 0E .. 9D F2 .. BB 88 .. 49 A6 1E 3B C0 CD .. : 1 times
0323: CB 87 E0 C6 80 68 72 A8 71 A9 71 B7 E9 45 F1 C3 : 1 times
0324: 94 4B DB 7C .. 8D 69 54 3A 55 82 76 AB A1 49 8B : 1 times
0325: 4D 5F F9 C0 A2 AD AE 6C 60 E3 EA B3 EC CC 83 D3 : 1 times
0326: 25 61 2A 1F A0 83 B9 20 AE 6F 6E .. 63 1C E2 C6 : 1 times
0327: 21 45 76 84 09 C3 AB 34 22 EB EE 1C EB 8E A6 87 : 3 times
                                                    (TOTAL 412)
```

F.2 Fautes obtenues durant la perturbation EM de l'implémentation matérielle de l'AES

F.2.1 Perturbations avec l'injecteur EM Cyl.-∅1500-N5

```
------------------------ REFERENCES --------------------------
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (Message)
3B E3 22 66 2F 3B E8 41 50 2E 79 41 46 05 25 49 ( Key )
52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 (Cipher )
---------------------------- RESUME ----------------------------
0000: .. .. .. .. .. .. .. .. E0 .. .. .. D6 .. .. .. .. : 1 times
0001: .. .. .. .. .. .. .. .. 10 .. .. .. .. .. .. .. .. : 6 times
0002: .. .. A4 00 70 01 .. .. .. .. 8D .. .. .. .. .. : 1 times
0003: .. .. .. .. 70 01 00 10 .. .. .. .. .. .. .. .. : 1 times
0004: 70 84 F6 D7 A1 A5 71 74 15 EA 46 6C 75 C1 DC 94 : 1 times
0005: .. .. .. .. .. .. .. .. 20 .. .. .. 16 .. .. .. .. : 1 times
0006: 00 00 .. .. 70 01 .. .. 00 00 .. .. 00 00 .. .. : 1 times
0007: D8 E6 D4 F1 74 C4 45 CF 9A 7B FF 08 B3 D8 4F 9C : 1 times
0008: .. .. A4 00 70 01 .. .. .. .. 8E .. .. .. .. .. : 1 times
                                                    (TOTAL 14)
```

F.2.2 Perturbations avec l'injecteur EM Cyl.-∅800-N7

```
------------------------- REFERENCES --------------------------
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (Message)
3B E3 22 66 2F 3B E8 41 50 2E 79 41 46 05 25 49 ( Key )
52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 (Cipher )
---------------------------- RESUME ---------------------------
0000: C1 D7 A7 A3 7E 14 CC 1B 28 DD C4 89 01 73 2E E4 : 1 times
0001: ED EE 77 6F B4 02 DC FD D0 12 E6 A1 38 5A 59 CB : 1 times
0002: 10 4D A4 00 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0003: E5 CD 2D 54 3F 71 57 91 90 E4 22 40 1A 3D .. 4E : 1 times
0004: .. 4D E4 84 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0005: 10 .. B0 10 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0006: 00 05 00 04 03 45 8E 20 .. .. .. .. .. .. .. .. : 2 times
0007: .. 4D .. .. .. .. .. .. .. .. .. .. .. .. .. .. : 4 times
0008: .. 4D .. 8C .. .. .. .. .. .. .. .. .. .. .. .. : 3 times
0009: 7C 0F 77 3E DC A7 7A BA C3 36 75 9C 52 21 34 0C : 1 times
0010: CC .. CC CC 70 .. 00 10 CC .. CC CC CC CC CC CC : 1 times
0011: .. .. .. .. .. .. .. .. .. .. AD 89 .. .. .. .. : 1 times
0012: A0 92 FF 8C 2D 66 97 C2 DC C6 69 C1 F4 1E BA F0 : 1 times
0013: 0F F5 88 64 27 68 FA 8F C7 37 9B 0B 8C 3F 31 AF : 2 times
0014: B9 36 10 43 18 C7 73 40 F1 A1 16 2A 97 B1 C6 C7 : 1 times
0015: .. .. .. .. .. .. .. .. .. .. AD .. .. .. .. .. : 3 times
0016: .. .. .. 1C .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0017: FB 90 FB CD .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0018: 12 4D A4 84 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0019: .. 4D .. 84 .. .. .. .. .. .. .. .. .. .. .. .. : 2 times
0020: 92 E2 45 59 7A C6 72 41 1D C0 D6 5E 44 AB F0 21 : 1 times
0021: .. .. .. .. .. .. .. .. .. B9 AD 8E .. .. .. .. : 1 times
0022: AF 70 BB 63 FD 36 CA 43 F9 85 8A 40 95 DF 5B E9 : 2 times
0023: 5A A1 66 26 CD 25 2A 57 07 B8 92 F6 44 20 DE F5 : 22 times
0024: .. 49 .. .. .. .. .. .. .. .. .. .. .. .. .. .. : 3 times
0025: 00 00 00 00 01 04 08 00 .. .. .. .. .. .. .. .. : 2 times
0026: .. .. .. .. .. .. .. .. .. 4D 5F 5D F0 .. .. .. : 1 times
0027: 2D 26 62 53 5F 9F FF 81 DF 41 E1 67 1F 64 .. 38 : 1 times
0028: CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC : 143 times
0029: CE 07 6B 37 4C D1 0A DB D4 FF A4 80 65 BF ED 98 : 1 times
0030: C6 14 79 F4 D9 3E 3E 52 0A 5B 22 D6 42 B7 EC E7 : 1 times
0031: 00 4D A4 00 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0032: 07 77 6F CA E6 A8 69 82 CE 37 B6 B1 5E 23 00 6B : 12 times
0033: 14 13 6F E5 87 1E DA 69 3E CE A4 E1 50 06 4C 5A : 1 times
                                                    (TOTAL 227)
```

F.2.3 Perturbations avec l'injecteur EM Ω-∅1800.450-N6

F.2.3.1 Orientations $\varphi = 0$

```
------------------------- REFERENCES --------------------------
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (Message)
3B E3 22 66 2F 3B E8 41 50 2E 79 41 46 05 25 49 ( Key )
52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 (Cipher )
---------------------------- RESUME ---------------------------
0000: CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC : 5 times
0001: DA C8 C7 BB 8C A7 EC 32 9E 51 30 FE C2 87 97 CC : 1 times
0002: 5A A1 66 26 CD 25 2A 57 07 B8 92 F6 44 20 DE F5 : 8 times
                                                        (TOTAL 14)
```

F.2.3.2 Orientations $\varphi = \frac{\pi}{2}$

F.2.3.2.1 Balayage global

```
 ------------------------- REFERENCES --------------------------
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (Message)
3B E3 22 66 2F 3B E8 41 50 2E 79 41 46 05 25 49 ( Key )
52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 (Cipher )
---------------------------- RESUME ---------------------------
0000: E4 1D 02 38 AA EA 7B C1 41 36 61 51 47 FD 0F 55 : 1 times
0001: 06 6C 7E B7 41 0D 65 48 01 73 41 75 E7 69 8F 58 : 1 times
0002: CC CC 2E CC .. .. .. .. CC 45 80 20 .. .. .. .. : 2 times
0003: 00 01 00 04 03 45 8E 20 .. .. .. .. .. .. .. .. : 1 times
0004: 42 .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. : 4 times
0005: 3A 00 4D A1 20 B2 BF 9D 54 CC 86 A8 CC AD F3 39 : 1 times
0006: 72 4D .. .. .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0007: .. .. .. .. 51 20 CC CC .. .. .. .. .. .. .. .. : 2 times
0008: 82 C7 C1 DE 3C C3 90 1B 08 39 38 D1 15 E7 C8 F4 : 1 times
0009: .. .. .. .. .. .. .. .. 00 09 00 .. 01 44 8E .. : 1 times
0010: .. .. .. .. 52 4F F4 9C .. .. .. .. .. .. .. .. : 2 times
0011: 38 EB 9C 91 56 F6 08 C9 6D AE E0 F5 E2 8F 02 B6 : 8 times
0012: 72 02 A7 E5 5E CC 6D 7E 04 72 A3 D5 79 A0 0B 7C : 2 times
0013: .. .. .. .. .. .. .. .. B1 46 9E 13 00 00 00 00 : 2 times
0014: .. .. .. .. 77 93 AF 9C .. .. .. .. .. .. .. .. : 1 times
0015: 00 4D 00 04 03 .. 8E .. .. .. .. .. .. .. .. .. : 1 times
0016: CC CC CC CC .. .. .. .. .. .. .. .. .. .. .. .. : 5 times
0017: E4 23 99 FF 2B 20 97 81 C3 CF 7A 1E 7E 71 D6 3F : 1 times
0018: 93 FB 05 F9 14 0D D7 8B 1C 95 C5 14 AF 52 4F 9B : 1 times
0019: E5 56 F0 B7 D0 DA 55 9D 9D 87 E8 D1 87 22 3E 4A : 1 times
0020: .. .. .. .. .. .. .. .. B1 46 9E 13 .. .. .. .. : 46 times
0021: C4 E7 3A 69 68 AD 3B 2C 9C 8D 5D 58 D6 DA 66 43 : 1 times
0022: 5A A1 66 26 CD 25 2A 57 07 B8 92 F6 44 20 DE F5 : 1 times
```

```
0023: 32 AE 6B 32 F8 4C 8B 3E 14 3A FC 4A 13 D6 0F 86 : 1 times
0024: .. .. .. .. 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 : 1 times
0025: 70 01 00 10 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 : 17 times
0026: .. .. .. .. .. .. .. .. CC 45 80 20 .. .. .. .. : 2 times
0027: .. .. .. .. D1 20 CC CC .. .. .. .. .. .. .. .. : 1 times
0028: C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 .. .. .. .. : 1 times
0029: .. .. .. .. .. 45 .. .. .. .. .. .. .. .. .. .. : 1 times
0030: .. .. .. .. .. .. .. .. C3 C5 AE 60 B8 A9 81 56 : 6 times
0031: CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC : 51 times
0032: 12 .. .. .. 03 .. .. .. .. .. .. .. .. .. .. .. : 2 times
0033: .. .. .. .. .. .. .. .. .. .. .. .. B8 A9 81 56 : 7 times
0034: 70 01 00 10 .. .. .. .. .. .. .. .. .. .. .. .. : 8 times
0035: .. .. .. .. .. .. .. .. FB 53 A5 5D .. .. .. .. : 1 times
0036: CC CC CC CC 50 CC CC CC .. .. .. .. .. .. .. .. : 3 times
0037: .. .. .. .. .. .. .. .. .. .. .. .. 10 44 BE .. : 1 times
                                                    (TOTAL 190)
```

F.2.3.2.2 Balayage sur la Zone 1

```
 ------------------------- REFERENCES --------------------------
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (Message)
3B E3 22 66 2F 3B E8 41 50 2E 79 41 46 05 25 49 ( Key )
52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 (Cipher )
---------------------------- RESUME ----------------------------
0000: 80 7F 4C 24 7A 1E C0 43 C9 1A 0A 90 C2 2B 63 66 : 1 times
0001: A2 3A DF 1B 0B 70 09 7E 2D 03 C3 35 19 80 12 5C : 1 times
0002: C3 C5 AE 60 .. .. .. .. B1 46 9E 13 00 00 00 00 : 1 times
0003: CA .. .. .. .. .. .. 2A .. .. 1D .. .. 1A .. .. : 1 times
0004: DC 49 55 11 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0005: 29 5D 88 B5 64 7F 98 76 14 A5 78 03 CA 95 42 DA : 2 times
0006: E9 E5 26 AA 35 DD 30 AF FA 5E 41 19 89 7F AE 31 : 1 times
0007: .. .. .. .. .. .. .. .. .. .. .. .. 11 .. BE .. : 1 times
0008: .. .. .. .. 13 93 72 66 .. .. .. .. .. .. .. .. : 1 times
0009: 82 C7 C1 DE 3C C3 90 1B 08 39 38 D1 15 E7 C8 F4 : 5 times
0010: .. .. .. .. 52 4F F4 9C .. .. .. .. .. .. .. .. : 45 times
0011: .. .. .. .. D1 20 00 CC .. .. .. .. .. .. .. .. : 2 times
0012: .. .. .. .. D1 20 CC CC .. .. .. .. .. .. .. .. : 5 times
0013: .. .. .. .. .. .. .. .. 10 09 00 .. 01 44 8E .. : 1 times
0014: 14 13 6F E5 87 1E DA 69 3E CE A4 E1 50 06 4C 5A : 1 times
0015: C8 19 4B 9D 94 52 5C AB C5 0E EA EC F2 A8 BC CA : 1 times
0016: .. .. .. .. 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 : 37 times
0017: 94 35 92 F7 1C 80 2E D9 8A 69 60 5B 9A 09 10 DF : 1 times
0018: 9F 52 44 8A 28 49 36 A7 67 AC 74 42 B9 3C 38 47 : 1 times
0019: 06 6C 7E B7 41 0D 65 48 01 73 41 75 E7 69 8F 58 : 2 times
0020: .. .. .. .. .. .. .. .. .. .. AB .. 16 .. .. .. : 1 times
```

```
0021: 7B E8 C3 AF 61 95 4F 89 F3 6D 45 BF 73 9E CD 2B : 1 times
0022: A9 A5 CB 63 66 EB 0A 67 E6 4B 59 AC AE 8C A9 A7 : 1 times
0023: .. .. .. .. .. .. .. .. CC 45 80 20 .. .. .. .. : 8 times
0024: .. .. .. .. 00 CC CC CC .. .. .. .. .. .. .. .. : 5 times
0025: 07 77 6F CA E6 A8 69 82 CE 37 B6 B1 5E 23 00 6B : 3 times
0026: .. 0D E4 .. .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0027: .. .. .. .. .. .. .. .. .. .. .. .. .. .. BE .. : 4 times
0028: .. .. .. .. .. .. .. .. 6A 69 A5 95 .. .. .. .. : 1 times
0029: CC CC 2A CC .. .. .. .. CC 45 80 20 .. .. .. .. : 3 times
0030: .. .. .. .. .. .. .. .. .. .. .. .. 11 44 BE .. : 6 times
0031: 2B F0 4F 45 B5 E1 1D FC 52 FF 55 88 14 4E 54 AF : 3 times
0032: F2 .. 57 8C .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0033: D4 8E C3 5A 8B BB 17 65 0C 2F 50 A5 3B D8 20 6F : 1 times
0034: 6E .. C5 FC .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0035: .. .. .. .. .. .. .. .. .. .. .. .. 11 44 AA 12 : 1 times
0036: .. .. .. .. 51 20 CC CC .. .. .. .. .. .. .. .. : 3 times
0037: CC CC CC CC 10 CC CC CC .. .. .. .. .. .. .. .. : 7 times
0038: .. .. .. .. .. .. .. .. .. .. .. .. C3 C5 AE 60 : 1 times
0039: 93 FB 05 F9 14 0D D7 8B 1C 95 C5 14 AF 52 4F 9B : 8 times
0040: E5 56 F0 B7 D0 DA 55 9D 9D 87 E8 D1 87 22 3E 4A : 4 times
0041: 6A .. C4 EC .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0042: BC 97 72 FE 0F 14 55 E9 72 A3 59 96 8F 0C 81 6E : 1 times
0043: .. .. .. .. CD 42 76 78 .. .. .. .. .. .. .. .. : 1 times
0044: A6 E0 59 E5 A0 1E E9 28 DF AB C0 D5 EE 7C 29 0D : 5 times
0045: C3 C5 AE 60 B8 A9 81 56 B1 46 9E 13 .. .. .. .. : 10 times
0046: 4A B6 74 E5 F3 07 03 DE BD 9B FA 4F AC 63 B1 A8 : 1 times
0047: .. .. .. .. .. .. .. .. .. .. .. .. B8 A9 81 56 : 338 times
0048: .. .. .. .. 03 .. .. .. .. .. .. .. .. .. .. .. : 9 times
0049: E4 1D 02 38 AA EA 7B C1 41 36 61 51 47 FD 0F 55 : 15 times
0050: .. .. .. .. .. .. .. .. .. .. 29 .. .. B8 A9 81 56 : 1 times
0051: .. .. .. .. .. .. .. .. .. .. 00 00 00 00 .. .. .. .. : 5 times
0052: .. .. .. .. .. .. .. .. .. .. .. .. .. .. 00 00 00 00 : 1 times
0053: 68 54 B0 B3 67 AD E1 54 F3 17 49 FB 94 97 2A EB : 2 times
0054: CC CC CC CC 00 CC CC CC .. .. .. .. .. .. .. .. : 1 times
0055: 21 E2 3C 86 24 1A AC F4 42 EA 60 E0 BC C7 C5 9C : 1 times
0056: .. .. .. .. .. .. .. .. .. .. .. .. 15 A2 41 73 : 1 times
0057: 72 02 A7 E5 5E CC 6D 7E 04 72 A3 D5 79 A0 0B 7C : 6 times
0058: .. .. .. .. .. .. .. .. B0 5B 22 87 .. .. .. .. : 1 times
0059: 57 .. F6 FE .. .. .. 56 .. .. .. .. 86 93 1E 17 : 1 times
0060: 49 47 8F CC 59 2F 65 D4 A0 7F 5E 9C F9 63 00 7B : 1 times
0061: E4 23 99 FF 2B 20 97 81 C3 CF 7A 1E 7E 71 D6 3F : 5 times
0062: .. .. .. .. .. .. .. .. 45 0E A5 12 .. .. .. .. : 1 times
0063: .. .. .. .. 50 00 CC CC .. .. .. .. .. .. .. .. : 2 times
0064: .. .. .. .. .. .. .. .. .. .. .. .. 10 44 BE .. : 6 times
0065: .. .. .. .. .. .. .. .. .. .. .. .. 00 00 00 CC : 1 times
```

```
0066: .. .. .. .. .. .. .. .. .. .. .. .. .. .. 81 BD C6 32 : 1 times
0067: 70 01 00 10 52 4F F4 9C C3 C5 AE 60 B8 A9 81 56 : 16 times
0068: .. .. .. .. .. .. .. .. CC C5 CC CC .. .. .. .. : 4 times
0069: .. .. .. .. .. .. .. .. EC D9 AC DD .. .. .. .. : 1 times
0070: 1A EB 84 F4 30 08 FE B0 E0 25 68 D4 7B 6E 39 6F : 8 times
0071: E5 41 D8 02 46 BC 18 18 EC 10 F7 54 0B 82 00 DE : 1 times
0072: 9A 46 7C EC 35 51 61 F1 EA 0A 82 2E 03 FF C9 24 : 1 times
0073: .. .. .. .. .. .. .. .. .. .. .. .. 89 97 00 7C : 1 times
0074: .. .. .. .. .. .. .. .. .. .. E9 .. .. B8 A9 81 56 : 1 times
0075: CC CC CC CC 50 CC CC CC .. .. .. .. .. .. .. .. : 11 times
0076: 41 9D 45 F0 C4 91 51 EE FE 4F C3 A9 04 C4 D1 53 : 1 times
0077: 87 DB A1 72 81 A8 95 F7 32 10 82 7D 83 09 F5 5D : 4 times
0078: .. .. .. .. .. .. .. .. 52 4F F4 9C B8 A9 81 56 : 8 times
0079: 07 C4 D5 8B F3 8D F9 56 0E 59 94 7B 31 A2 65 38 : 2 times
0080: C2 DF 5B D1 .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0081: 2A C5 29 04 47 F2 77 56 83 3C 0C 25 75 F6 47 BE : 1 times
0082: .. .. .. .. .. .. .. .. .. .. .. .. 10 44 AE 12 : 1 times
0083: CC CC CC CC 50 00 CC CC .. .. .. .. .. .. .. .. : 11 times
0084: 12 .. .. .. 03 .. .. .. .. .. .. .. .. .. .. .. : 6 times
0085: 38 EB 9C 91 56 F6 08 C9 6D AE E0 F5 E2 8F 02 B6 : 43 times
0086: .. .. .. .. .. .. .. .. .. .. AB .. .. .. .. .. : 1 times
0087: 72 C5 FC C6 94 70 DD A8 25 05 13 85 E1 38 6F 04 : 3 times
0088: CC CC 2E CC .. .. .. .. CC 45 80 20 .. .. .. .. : 3 times
0089: CC CC CC CC .. .. .. .. .. .. .. .. .. .. .. .. : 1 times
0090: .. .. .. .. .. .. .. .. .. .. .. .. 31 .. BE .. : 8 times
0091: C3 C5 AE 60 CC .. CC CC B1 46 9E 13 .. .. .. .. : 1 times
0092: .. .. .. .. .. .. .. .. B1 46 9E 13 .. .. .. .. : 25 times
0093: 23 62 EE FC 97 73 C8 FF 56 2C 2D 3F 09 CA AB 80 : 4 times
0094: .. .. .. .. .. .. .. .. 38 89 00 .. 21 44 .. .. : 2 times
0095: DD 14 00 84 1C AF 02 48 D9 3A 7A D1 A4 4A 1C 08 : 1 times
0096: .. .. .. .. .. 45 .. .. .. .. .. .. .. .. .. .. : 1 times
0097: C9 5F 02 FD A1 65 D7 0A 12 D3 3D 7C 5C 82 20 69 : 1 times
0098: .. .. .. .. .. .. .. .. C3 C5 AE 60 B8 A9 81 56 : 12 times
0099: CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC : 334 times
0100: .. .. .. .. 50 20 CC CC .. .. .. .. .. .. .. .. : 8 times
0101: .. .. .. .. 83 .. .. .. .. .. .. .. .. .. .. .. : 4 times
0102: 70 01 00 10 .. .. .. .. .. .. .. .. .. .. .. .. : 3 times
0103: 0A 12 D0 FA AB 2E 02 38 D2 F8 D0 82 54 03 D4 55 : 1 times
                                                    (TOTAL 1128)
```