Évaluation par simulation de la sécurité des circuits face aux attaques par faute

Olivier Faurax

UNIVERSITÉ DE LA MEDITERRANÉE
ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE E.D. 184

THÈSE

présentée pour obtenir le grade de
Docteur de l'Université de la Méditerranée
Spécialité : Informatique
par

Olivier FAURAX

sous la direction du Pr. Traian MUNTEAN

Titre :

Évaluation par simulation de la sécurité des circuits face aux attaques par faute

soutenue publiquement le 3 juillet 2008

JURY

M. Jean ARLAT	Rapporteur
M. Jean-Jacques QUISQUATER	Rapporteur
M. Traian MUNTEAN	Directeur de thèse
M. Frédéric BANCEL	Examinateur
M. Alexis BONNECAZE	Examinateur
M. Pierre PARADINAS	Examinateur
Mme. Assia TRIA	Examinateur

Remerciements

Je remercie tout d'abord Monsieur Pierre Paradinas pour l'intérêt qu'il a porté à mon travail en me faisant l'honneur d'être président du jury. Merci également à Monsieur Jean Arlat et Monsieur Jean-Jacques Quisquater d'avoir accepté de rapporter mon travail. C'est un réel honneur de faire juger son travail par des références incontestables dans leurs domaines.

J'adresse mes remerciements à Monsieur Traian Muntean pour avoir été mon directeur de thèse et pour la liberté de travail qu'il m'a laissé pendant ces trois années de thèse.

Toute ma gratitude va à l'équipe du Laboratoire SAS du site Georges Charpak à Gardanne où j'ai effectué ma thèse et notamment à Assia Tria, qui m'a de plus fait l'amitié d'être membre du jury.

Je tiens à remercier en particulier Jean-Baptiste (qui sait utiliser la force et son côté obscur), Pascal (dont la qualité scientifique n'a d'égal que son tact), Michel (qui m'a fait découvrir que le trac a parfois du bon), Marc (quelle est la probabilité qu'un nombre soit premier ?), Jérôme (qui devrait manger bio pour sa santé), Philippe (merci pour l'aide LaTeX !), ainsi que Laurent qui a suivi ma thèse au quotidien et ceux qui me pardonneront de les avoir oubliés.

J'adresse tous mes voeux de réussite à Julien (le meilleur candidat des JNRDM 2007), à Selma et à tous les autres stagiaires et thésards du laboratoire pour leurs travaux respectifs.

Ma profonde reconnaissance va au personnel du Centre Microélectronique de Provence Georges Charpak qui m'a épaulé pendant ces années et notamment Véronique, Christelle et François (compagnon de galère !), sans oublier Philippe Collot, Directeur du centre, pour toute l'aide logistique mais aussi morale dont j'ai bénéficié.

J'adresse un grand merci à l'équipe ERISCS de Luminy et au personnel de l'ESIL. Je souhaite bonne continuation à Hervé, Victor, Arnaud dans leur voies respectives et je remercie tout particulièrement Laurent pour son aide précieuse. J'ai également apprécié discuter avec Robert Rolland et Alexis Bonnecaze qui m'a fait l'honneur de participer à mon jury.

Cette thèse a été réalisée en collaboration avec le département SmartCard de

STMicroelectronics à Rousset où j'ai apprécié le soutien d'Anissa, de Nicolas et de toute l'équipe. Toute ma reconnaissance va à Frédéric Bancel pour avoir toujours été présent pour m'aider et m'avoir fait l'amitié d'être membre du jury.

Je n'aurai jamais de mots assez chaleureux pour exprimer toute ma gratitude à ma mère, mon père et Jérémie, mon frère, pour leur affection, leur dévouement et leur soutien de chaque instant. Cette thèse leur est dédiée.

Ma pensée va également à mon parrain, ma grand-mère et tous les membres de ma famille qui m'ont encouragé pendant ma thèse.

Merci aussi à tous ceux qui ont réussi à me faire sortir de ma thèse de temps en temps, notamment Rémi (qui cherche l'aventure dans la vie et sur le net), l'Harmonie Municipale d'Aix-en-Provence (particulièrement Julien, Hervé, Yannick et Noémie), mais aussi les espérantistes d'Aix-en-Provence et d'ailleurs (Sylvaine, Parŝo, Mireille, Adeline, Vincent et JEFO).

Je n'oublie pas non plus tous les gens derrière les projets libres qui m'ont facilité la vie : Mandriva GNU/Linux (#mandrivafr, mandriva@jabberfr.org), Wikipedia, LaTeX (et Beamer) et Wilson Snyder pour Verilog-Perl (sur lequel PAFI est basé).

Pour finir, je te remercie, lecteur, de lire les remerciements jusqu'à la fin. Je sais très bien que cette ligne sera plus lue que la plupart des lignes de cette thèse alors maintenant que tu es arrivé jusqu'ici, fais-moi une faveur : lis le résumé de la page suivante.

Résumé

Les circuits microélectroniques sécuritaires sont de plus en plus présents dans notre quotidien (carte à puce, carte SIM) et ils renferment des informations sensibles qu'il faut protéger (numéro de compte, clé de chiffrement, données personnelles).

Récemment, des attaques sur les algorithmes de cryptographie basées sur l'utilisation de fautes ont fait leur apparition. L'ajout d'une faute lors d'un calcul du circuit permet d'obtenir un résultat faux. À partir d'un certain nombre de résultats corrects et de résultats faux correspondants, il est possible d'obtenir des informations secrètes et dans certains cas des clés cryptographiques complètes.

Cependant, les perturbations physiques utilisées en pratique (impulsion laser, radiations, changement rapide de la tension d'alimentation) correspondent rarement aux types de fautes nécessaires pour réaliser ces attaques théoriques.

Dans ce travail, nous proposons une méthodologie pour tester les circuits face aux attaques par faute en utilisant de la simulation. L'utilisation de la simulation permet de tester le circuit avant la réalisation physique mais nécessite beaucoup de temps. C'est pour cela que notre méthodologie aide l'utilisateur à choisir les fautes les plus importantes pour réduire significativement le temps de simulation.

L'outil et la méthodologie associée ont été testés sur un circuit cryptographique (AES) en utilisant un modèle de faute utilisant des délais. Nous avons notamment montré que l'utilisation de délais pour réaliser des fautes permet de générer des fautes correspondantes à des attaques connues.

Mots-clés : DFA, attaque par faute, sécurité, informatique, cryptographie, microélectronique, simulation, AES, carte à puce

Resumo

Mikroelektronikaj cirkvitoj uzataj por sekureco pli kaj pli ĉeestas en nia ĉiutaga vivo (memorkarto, SIM karto) kaj ili enhavas gravajn informojn kiujn necesas protekti (numero de konto, ĉifroŝlosilo, personaj datumoj).

Lastatempe, aperis atakoj kontraŭ ŝlosiloalgoritmoj, kiuj uzas erarojn. Aldono de eraro dum komputado de cirkvito ebligas miskomputitajn rezultojn. Kun korektaj rezultoj kaj korespondantaj miskomputitaj rezultoj, eblas malkovri sekretajn informojn kaj, en kelkaj kazoj, kompletajn ĉifroŝlosilojn.

Tamen, fizikaj perturboj uzataj praktike (lasero, radiaĵo, energia intermito) ofte ne kongruas kun bezonitaj eraroj por realigi tiujn teoriajn atakojn.

En tiu laboro, ni proponas metodologion por testi cirkvitojn kontraŭ atakoj kiuj uzas erarojn, per simulado. Uzi simuladon ebligas testi la cirkviton antaŭ ĝia fizika realigo sed bezonas multe da tempo. Tial nia metodologio helpas la uzanton elekti la plej gravajn erarojn por redukti la daŭron necesan.

La ilo kaj la rilata metodologio estis testitaj en ĉifrocirkvito (AES) uzante erarmodelon kiu aldonas malfruojn. Ni notinde demonstris ke uzi malfruojn por realigi erarojn ebligas krei erarojn kiuj korespondas al konataj atakoj.

Ŝlosilvortoj : DFA, eraratako, sekureco, komputiko, kriptologio, mikroelektroniko, simulado, AES, inteligenta memorkarto

La nomo de la ilo estas PAFI, ĉar kiam ni aldonas erarojn en la cirkvito, estas same kiel pafi ĝin per lasero.

Abstract

Microelectronic security devices are more and more present in our lives (smartcards, SIM cards) and they contains sensitive informations that must be protected (account number, cryptographic key, personal data).

Recently, attacks on cryptographic algorithms appeared, based on the use of faults. Adding a fault during a device computation enables one to obtain a faulty result. Using a certain amount of correct results and the corresponding faulty ones, it is possible to extract secret data and, in some cases, complete cryptographic keys.

However, physical perturbations used in practice (laser, radiations, power glitch) rarely match with faults needed to successfully perform theoretical attacks.

In this work, we propose a methodology to test circuits under fault attacks, using simulation. The use of simulation enables to test the circuit before its physical realization, but needs a lot of time. That is why our methodology helps the user to choose the most important faults in order to significantly reduce the simulation time.

The tool and the corresponding methodology have been tested on a cryptographic circuit (AES) using a delay fault model. We showed that use of delays to make faults can generate faults suitable for performing known attacks.

Keywords : DFA, fault attack, security, computer science, cryptography, microelectronics, simulation, AES, smartcard

Table des matières

Liste des acronymes

AES Advanced Encryption Standard
ASIC Application Specific Integrated Circuit
CPU Central Processing Unit
DCSSI Direction Centrale de la Sécurité des Systèmes d'Information
DEMA Differential ElectroMagnetic Analysis
DES Data Encryption Standard
DFA Differential Fault Analysis
DPA Differential Power Analysis
FPGA Field-Programmable Gate Array
ISO International Organization for Standardization
JTAG Joint Test Action Group
LUT LookUp table
NIST National Institute of Standards and Technology
PAFI Prototype of Another Fault Injector
RSA Rivest, Shamir, Adleman
RTL Register Transfer Level
SCIFI Scan Chain Implemented Fault Injection
SCQL Structured Card Query Language
SDF Standard Delay Format
SET Single Event Transient
SEU Single Event Upset
SIM Subscriber Identity Module
SWIFI SoftWare Implemented Fault Injection
USB Universal Serial Bus
VHDL VHSIC Hardware Description Language
VHSIC Very-High-Speed Integrated Circuits
VLSI Very-Large-Scale Integration

Introduction

La sécurité constitue une composante cruciale des technologies de l'information et de la communication car elle est à la base de l'instauration de la confiance nécessaire pour les utilisateurs finaux.

L'une des plus importantes menaces qui pèsent sur la sécurité est la vulnérabilité du matériel électronique qui implémente des fonctions de cryptographie, notamment celles de confidentialité, d'identification et d'authentification.

En effet, certaines manipulations frauduleuses ou « attaques » sur ce matériel permettent d'extraire des informations confidentielles comme les clefs de chiffrement, et ainsi de mettre à mal toute la chaîne de transmission sécurisée de l'information.

Comme la course-poursuite engagée entre les concepteurs de circuits sécuritaires et les attaquants s'accélère avec la diversité des systèmes, leur ouverture et leur multiplicité, il apparaît aujourd'hui comme un enjeu majeur dans la sécurisation des systèmes de communication, de devoir améliorer drastiquement la résistance des composants à ces techniques d'attaques.

La sécurité physique des composants est encore majoritairement dédiée au segment de la carte à puce, un marché de 4,2 milliards d'euros en 2005, dominé par des acteurs français.

Ce marché subit actuellement de profondes mutations : très forte pression concurrentielle, complexité et innovations constantes et structurantes qui forcent les acteurs à toujours plus d'efficacité en R&D et production avancée. Cette complexité pousse également les émetteurs de cartes à puces à imposer aux fabricants des processus de certification sécuritaires de plus en plus contraignants et qui ont un impact financier significatif.

La sécurité physique tend également à migrer vers tous les segments de l'électronique grand public avec l'évolution des applications et des usages. Ce marché est évalué à plus de 1200 milliards d'euros en 2005 dont 0,1% à 0,5% pour la sécurité, soit 1 à 5 milliards d'euros. Ce nouveau segment fait donc l'objet d'un enjeu majeur pour les industriels de la sécurité numérique.

Les circuits intégrés dédiés aux applications de sécurité contiennent et manipulent des données privées et/ou critiques. Ces secrets embarqués doivent être protégés pour ne pas être extraits par des personnes non-autorisées.

Les industriels cherchent à miniaturiser au maximum leur circuits, ce qui les rend plus sensibles aux perturbations physiques, notamment dans le domaine de l'aéronautique où les radiations cosmiques peuvent affecter des données. À haute altitude, des observations ont mis en évidence des modifications d'état dans des mémoires SRAM provoquées par des neutrons. Des fautes de ce type ont été reproduites en laboratoire par radiation [OBF+93].

Ces radiations ont aussi de plus en plus un impact sur d'autres applications critiques embarquées terrestres (automobile, ferroviaire, etc.). La protection contre ces erreurs est du domaine de la sûreté de fonctionnement, où on cherche à assurer le fonctionnement correct du circuit face à des fautes physiques aléatoires.

Les circuits de cartes à puces sont la cible d'attaques de plus en plus sophistiquées comme les attaques par faute qui combinent perturbations physiques et cryptanalyse. Ces attaques consistent à perturber physiquement une ou plusieurs parties d'un circuit en vue de le corrompre et d'en exploiter des résultats erronés. Le principe est d'exécuter des calculs inter-dépendants à une faute près, permettant d'extraire par cryptanalyse différentielle des données protégées.

Notre problématique est proche de celle de la sûreté de fonctionnement. Cependant, un attaquant va essayer d'orienter ses fautes afin d'obtenir le plus rapidement et le plus facilement les informations qu'il souhaite, plutôt que d'injecter des fautes aléatoires. C'est aussi lui qui choisit le type de faute, alors que le type de faute est connu suivant l'utilisation du circuit (aéronautique, espace) dans le domaine de la sûreté de fonctionnement. En conséquence, bien que la partie technique (comme les outils d'injection de fautes) soit relativement similaire, il est nécessaire de proposer une nouvelle méthodologie et des critères adaptés d'évaluation du niveau de sécurité des circuits.

Pour concevoir des circuits robustes, la problématique de la sécurité doit être prise en compte très tôt dans le flot de conception des circuits, dans le but de garantir, dès l'étape de spécification fonctionnelle, l'obtention des certifications de sécurité nécessaires à leur mise sur le marché, délivrées par des organismes indépendants d'évaluation de la sécurité.

La conception de circuit utilise une approche « top-down ». Les circuits sont d'abord spécifiés fonctionnellement et vérifiés en simulation. Les circuits sont alors synthétisés vers une modélisation très proche du circuit physique final, puis vérifiés en utilisant des modèles de composants physiques fournis par les fondeurs. Pour prédire la sécurité des circuits le plus tôt possible, il faut utiliser des méthodologies de conception et des outils adaptés qui s'intègrent dans ce flot de conception.

Cette thèse vise à fournir une prédiction de la sécurité des circuits en amont dans ce flot de conception à travers une méthodologie originale.

La premier chapitre est consacré à la présentation des notions de base sur la sécurité, la cryptographie et les outils d'injection de fautes. Nous commençons par exposer les contraintes industrielles des circuits dédiés à la sécurité, notamment celles qui s'appliquent aux cartes à puces. L'état de l'art sur la cryptographie présente les principaux algorithmes ainsi que les attaques connues sur les circuits sur lesquels ils sont implémentés. Pour conclure ce premier chapitre, les avantages et inconvénients des méthodes d'injection de fautes sont présentés à travers les particularités des principaux outils, développés principalement dans le domaine de la sûreté de fonctionnement.

Le second chapitre expose les objectifs de la thèse par rapport aux travaux existants, puis notre méthodologie globale d'évaluation de la sécurité. La terminologie relative à la sûreté de fonctionnement est rappelée pour ensuite détailler les différents modèles de faute, ainsi que les modélisations numériques équivalentes. Comme le nombre de fautes potentielles est grand, il est nécessaire de proposer des critères de choix des fautes à simuler, en accord avec les objectifs de notre analyse. Pour évaluer la sécurité des circuits, il faut ensuite définir des métriques d'évaluation de la sécurité des circuits.

Le troisième chapitre présente l'implémentation de la méthodologie du précédent chapitre à travers l'outil PAFI (*Prototype of Another Fault Injector*). L'accent est mis sur le flot de conception de l'outil qui peut être divisé en 3 étapes : l'analyse préalable du circuit, la simulation des fautes sélectionnées et enfin l'analyse des résultats de simulation.

Le quatrième chapitre décrit les résultats d'expérimentation pour, dans un premier temps, estimer les performances de l'approche choisie et de l'outil associé sur une implémentation de l'algorithme de cryptographie AES. Dans un second temps, les niveaux de sécurité de différentes implémentations de l'AES sont comparés, en prenant en compte le modèle de faute de délai conjointement aux attaques par faute connues.

En conclusion, les évolutions possibles et les perspectives de ces travaux sont présentées. Une justification expérimentale des modèles de faute physiques serait le premier pas vers l'utilisation des fautes de délai pour des attaques concrètes. La méthodologie pourra être enrichie de nouveaux modèles de fautes correspondant à d'autres perturbations physiques. De plus, l'évaluation pourra être proposée sur d'autres circuits de sécurité, comme les circuits de cryptographie asymétrique, ou sur des circuits utilisant des contremesures.

Cette thèse a été réalisée au Centre Microélectronique de Provence Georges Charpak, à l'Université de la Méditerranée (Aix-Marseille 2) et en collaboration avec la division Smartcard de ST Microelectronics à Rousset.

L'équipe Systèmes et Architectures Sécurisées (SAS) du Centre de Microélectronique de Provence Georges Charpak (CMPGC) est située à Gardanne et effectue des recherches sur la conception sécurisée des systèmes embarqués et des circuits microélectroniques.

L'équipe Systèmes Informatiques Communicants est dans les locaux de l'École Supérieure d'Informatique de Luminy, dans le pôle scientifique de Luminy à Marseille.

Cette thèse a été financée grâce au plan Rousset 2003-2008 et a été réalisée en collaboration avec la division Smartcard de ST Microelectronics, située à Rousset, qui conçoit les circuits microélectroniques des cartes à puces du fondeur.

Chapitre 1

État de l'art sur la sécurité et l'injection de fautes

Ce travail de thèse se place à l'intersection de plusieurs domaines : la sécurité, la cryptographie et l'aide à la conception de circuits.

Dans un premier temps, nous rappelons le contexte de sécurité et les contraintes de production industrielle pour situer notre état de l'art par rapport à notre problématique. La carte à puce embarque des circuits spécifiques dédiés principalement à des applications de sécurité (SIM, Carte Bleue) et doit se conformer à des standards tout en réduisant le coût induit par la sécurité. De plus, apporter une nouvelle méthodologie de conception des circuits nécessite de l'intégrer dans le flot existant pour faciliter l'intégration industrielle.

La cryptographie est l'art de rendre un message inintelligible à un tiers non-autorisé. Cela permet de cacher une information secrète, et ainsi d'obtenir de la sécurité en protégeant des données sensibles. Les principaux algorithmes de cryptographie sont présentés, notamment l'AES, qui sera le cas d'étude du chapitre 4. Les différents types d'attaques connues sont aussi exposés et en particulier les attaques par faute.

Enfin, les principaux outils d'injection de fautes sont décrits et regroupés par méthode d'injection, pour recenser leurs avantages et inconvénients par rapport à la problématique de la sécurité.

1.1 Contraintes industrielles des systèmes sécuritaires

La sécurité des circuits doit être assurée en utilisant des méthodes de protection puisqu'elle est nécessaire à l'obtention des certifications nécessaires à leur commercialisation. Ces certifications permettent d'assurer la sécurité des systèmes en imposant aux industriels le recours à des tests sécuritaires menés par des laboratoires indépendants.

De plus, certains systèmes ont besoin de se prémunir contre la fraude. On pense immédiatement au domaine bancaire, mais les opérateurs téléphoniques représentent un marché beaucoup plus vaste (en valeur comme en unités), notamment au travers des cartes prépayés et des cartes SIM pour la téléphonie mobile, comme on peut le voir sur le figure 1.1.

	2007	prévisions 2008	évolution
Télécom	2600	3000	15%
Finance	500	580	16%
Gouvernement, Santé	105	150	43%
Transport	15	30	100%
Télévision	70	80	14%
Sécurité d'entreprise	20	20	0%
Autres	15	15	0%
Total	3325	3875	17%

FIG. 1.1 – Marché de la carte à puce à microprocesseur, en million d'unités (source : eurosmart.com)

Même si aucun cadre de certification n'existe, la télé à péage est aussi un domaine où la sécurité est primordiale, dans la mesure où une fraude permettrait à des non-abonnés de profiter gratuitement des contenus payants. Arrivés récemment en France, les passeports biométriques sont équipés d'une carte à puce sans contact et doivent garantir la protection contre la lecture par un tiers, la réplication et la modification des informations personnelles contenues.

L'ajout de ces protections a une répercussion sur le coût de fabrication ou les performances du système. Dans un contexte industriel, un compromis doit être recherché pour assurer la sécurité du système à moindre coût. Par exemple, dans le contexte de la conception microélectronique, les protections possibles augmentent le temps de calcul et/ou la taille du circuit. Cela implique une baisse des performances et/ou un surcoût en matière première qui doit être justifié.

Cette section présente le type de système considéré (carte à puce) ainsi que les contraintes de conception industrielles.

1.1.1 Système de carte à puce

Une carte à puce est une carte rectangulaire en matière plastique, de quelques centimètres de côté et moins d'un millimètre d'épaisseur, portant au moins un circuit intégré capable de contenir de l'information.

Le circuit intégré (la puce, figure 1.2), peut contenir un microprocesseur (CPU) capable de traiter cette information, ou être limité à des circuits de mémoire non volatile et, éventuellement, un composant de sécurité (carte mémoire). Les cartes à puce sont principalement utilisées comme moyens d'identification personnelle (carte

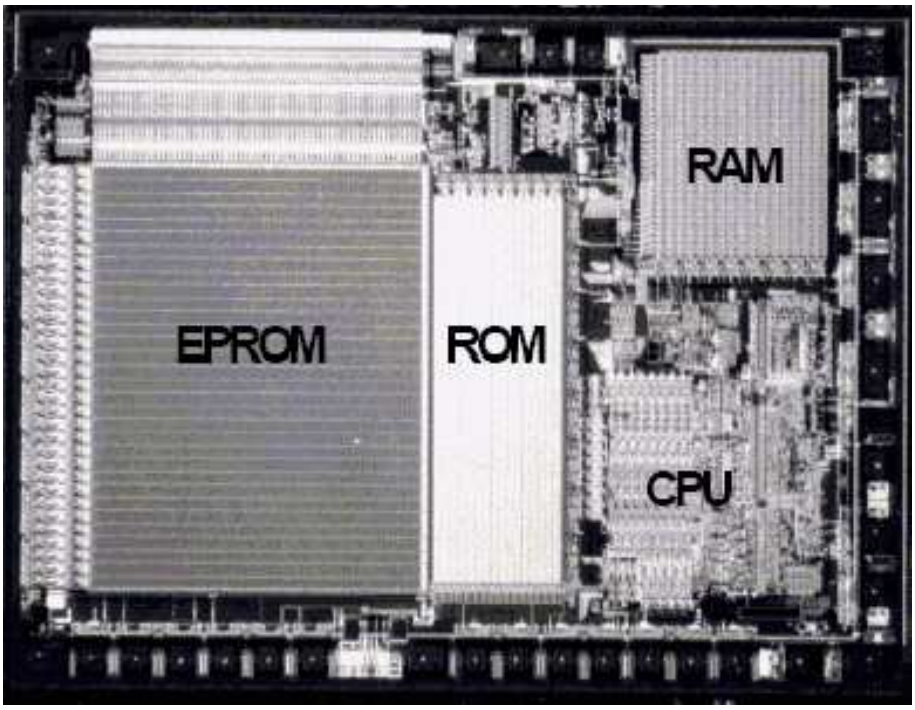

FIG. 1.2 – Circuit de la 1ère carte à puce, en Octobre 1981

d'identité, badge d'accès aux bâtiments, carte d'assurance maladie, carte SIM), de paiement (carte bancaire, porte-monnaie électronique) ou de preuve d'abonnement à des services prépayés (carte de téléphone, titre de transport, télévision à péage). La carte peut comporter un hologramme pour éviter la contrefaçon. La lecture, par des équipements spécialisés, peut se faire avec ou sans contact avec la puce.

Depuis son lancement commercial en 1992, la carte à puce a profité pleinement de la miniaturisation des circuits et, bien que la surface maximum de la puce soit standardisé à 25 mm^2, les circuits ont évolués vers plus de capacités mémoire et possibilités de calcul.

Les cartes à puces sont utilisées combinées à des terminaux comme les cartes bancaires, les cartes prépayées de téléphone, les cartes Vitale ou encore les cartes SIM, dont le terminal est le téléphone portable. Elles constituent la plupart du temps un maillon de la chaîne de sécurité d'un système plus vaste. Pour pouvoir intégrer ce composant sécuritaire, l'industrie a du définir des standards pour faciliter l'interopérabilité des équipements.

La famille de standards ISO/IEC 7816 définit les propriétés des cartes à puces avec contact. Les différents standards sont résumés dans l'annexe A. On retiendra notamment les standards ISO/IEC 7816-8 et -15 qui définissent les modalités d'utilisation des primitives de sécurité et l'accès aux fonctions cryptographiques de la carte, visibles sur la figure 1.3 (RSA, DES). Cette famille de standard montre qu'un système de carte à puce est soumis à un grand nombre de contraintes pour fournir

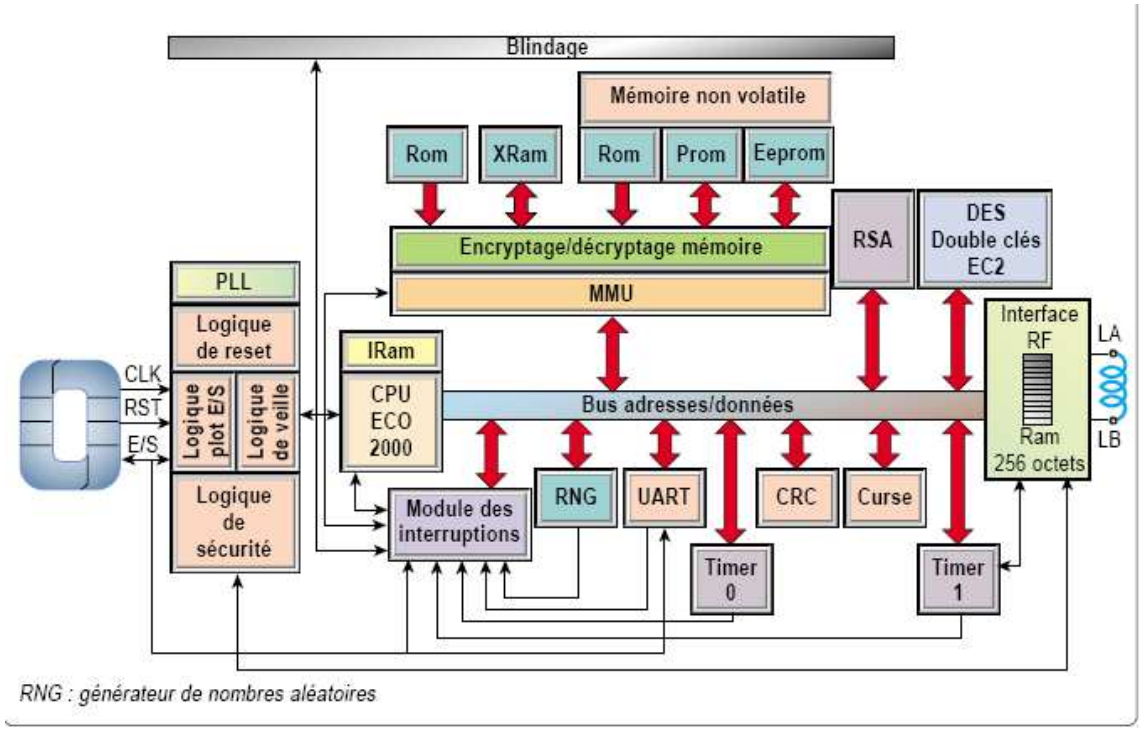

FIG. 1.3 – Architecture d'une carte à puce

un grand nombre de fonctionnalités :

- contraintes physiques : torsion possible, tolérance au rayonnement électromagnétique, 25 mm^2 de surface maximum
- sécurité : échange de messages sécurisés, accès aux fonctions de cryptographie
- politiques : gestion des droits d'accès, interopérabilité avec les terminaux

1.1.2 Contrainte de flot de conception

Pour faciliter l'intégration des outils développés dans le cadre de la thèse, il a été décidé d'utiliser les outils de l'industriel [1], c'est-à-dire NCSim (Cadence) pour le simulateur, et Design-Vision (Synopsys) pour l'outil de synthèse.

Cela implique l'utilisation d'un langage de description de circuit compatible, c'est-à-dire par exemple VHDL ou Verilog, pour pouvoir manipuler des circuits conçus dans un contexte industriel.

Une autre contrainte a été l'utilisation de la simulation parce que l'accès aux circuits physiques n'était pas possible pour des raisons de logistique et de protection de la propriété intellectuelle.

1.2 Cryptographie actuelle et attaques

La cryptographie est la science s'attachant à protéger des messages (assurant confidentialité, authenticité et intégrité) en utilisant des clés secrètes. Les premiers algorithmes étaient simplement des décalages cycliques des lettres des messages à chiffrer. La sécurité de ces algorithmes reposait plus sur le secret de l'algorithme que sur le nombre de décalage (la clé). Grâce à l'essor des mathématiques et à la puissance de calcul des ordinateurs actuels, les algorithmes sont maintenant beaucoup plus robustes, même lorsqu'ils sont publics. Cette section présente une rapide synthèse des principaux algorithmes actuels pour donner une vue d'ensemble des attaques dont ils sont les cibles.

1.2.1 Terminologie

À cause de l'utilisation d'anglicismes puis de la création des chaînes de télévision dites « cryptées », une grande confusion règne concernant les différents termes de la cryptographie.

Étymologiquement, **cryptographie** signifie « écriture secrète », devenue par extension l'étude de cet art (donc aujourd'hui la science visant à créer des cryptogrammes, c'est-à-dire à chiffrer).

[1] Cette thèse a été réalisée en collaboration avec la division SmartCard de STMicroelectronics Rousset.

Le **chiffrement** est la transformation à l'aide d'une clé de chiffrement d'un message en clair en un message incompréhensible si on ne dispose pas d'une clé de déchiffrement (en anglais *encryption*).

Le **cryptogramme** est le message chiffré.

Le **déchiffrement** est la transformation du cryptogramme vers le message en clair, en utilisant la clé de déchiffrement.

Décrypter, c'est retrouver le message clair correspondant à un message chiffré sans posséder la clé de déchiffrement (terme que ne possèdent pas les anglophones, qui eux « cassent » des codes secrets).

La **cryptanalyse** est la science analysant les cryptogrammes en vue de les décrypter. La cryptanalyse et la cryptographie forment la **cryptologie**.

Il apparaît donc que mis au regard du couple chiffrer/déchiffrer et du sens du mot « décrypter », le terme « crypter » n'a pas de raison d'être (l'Académie française précise que le mot est à bannir et celui-ci ne figure pas dans son dictionnaire), en tout cas pas dans le sens où on le trouve en général utilisé.

1.2.2 Cryptographie symétrique

On parle de cryptographie symétrique pour les algorithmes où la même clé secrète est utilisée pour chiffrer et déchiffrer les messages.

$$\textit{Cryptogramme} = \textit{Chiffrement(Clé,Message)}$$
$$\textit{Message} = \textit{Déchiffrement(Clé,Cryptogramme)}$$

C'est le schéma le plus ancien : celui qui envoie et celui qui reçoit utilisent le même secret, la clé, pour chiffrer et déchiffrer.

L'attaque la plus naïve consiste à essayer toutes les clés possibles jusqu'à obtenir un message plausible. Cela nécessite de connaitre au moins un texte chiffré et tout ou partie du message en clair correspondant. Pour rendre difficile cette recherche exhaustive, le nombre de clés possibles doit être très grand : on estime actuellement le minimum à 2^{80}, c'est-à-dire une sécurité de 80 bits.

En conséquence, une sécurité absolue peut être obtenue en utilisant une clé aléatoire de même taille que le texte à chiffrer : c'est le chiffrement de Gilbert Vernam [Ver26]. En effet, faire la bonne hypothèse de clé revient à définir le message à trouver.

Le principal désavantage de la cryptographie symétrique est que la clé doit faire partie du système de déchiffrement. Il faut protéger le système de chiffrement ainsi que celui de déchiffrement, sinon la clé peut être extraite et volée. Dans ce cas, l'attaquant peut ensuite espionner les échanges et même modifier les messages échangés.

1.2.2.1 DES et triple DES

DES est un ancien algorithme de chiffrement symétrique standardisé en 1977, puis révisé en 1988, 1993 et finalement 1998 [Sta99]. En 1999, le NIST (National

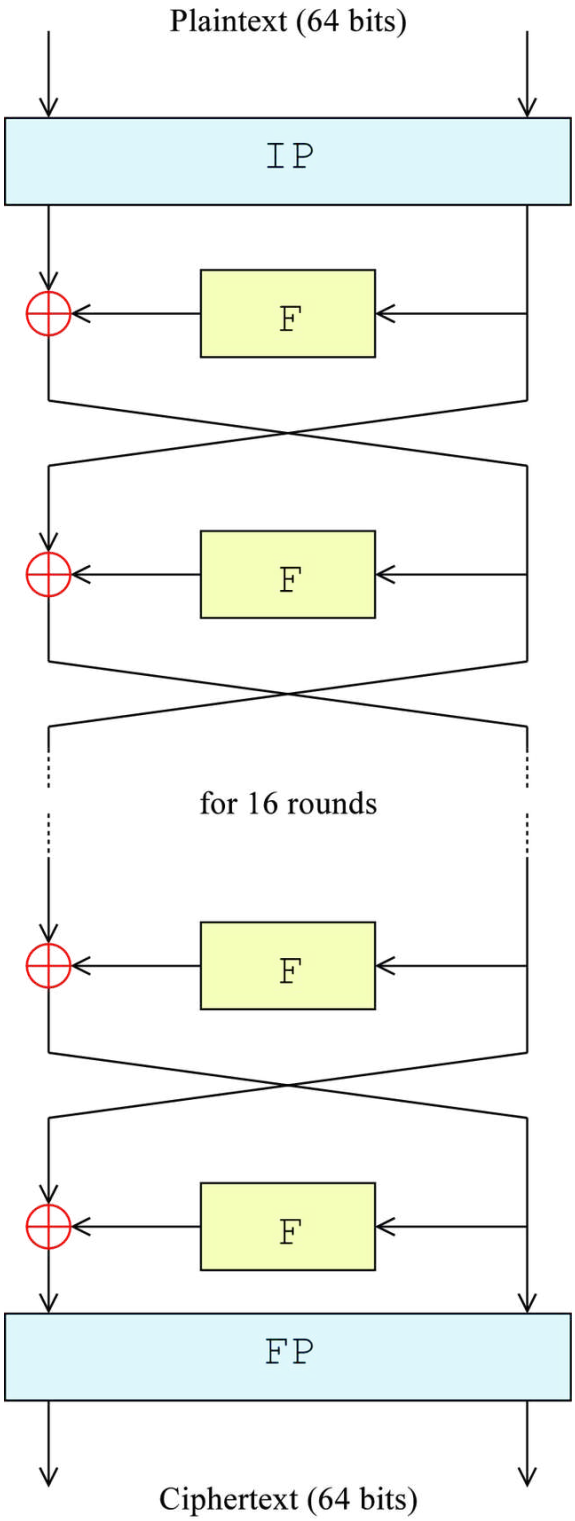

FIG. 1.4 – Structure générale du DES

Institute of Standards and Technology, institut de standardisation des États-Unis) recommande l'utilisation du Triple DES. Il sera ensuite remplacé comme standard officiel en 2001 par l'AES (présenté dans la section suivante), même s'il est toujours très utilisé aujourd'hui.

DES chiffre un texte clair de 64 bits avec une clé de 56 bits. L'algorithme prend cependant en entrée une clé de longueur 64 bits, mais 1 bit sur 8 est utilisé comme bit de parité. L'espace de clé est donc de 2^{56} ce qui est maintenant insuffisant contre une attaque par recherche exhaustive, compte tenu de la puissance de calcul disponible.

L'algorithme (figure 1.4) commence par une permutation IP des 64 bits de données d'entrée. Il effectue ensuite 16 rondes qui sont des schémas basés sur des cellules de Feistel (figure 1.5). L'algorithme se termine par l'application de la permutation FP, qui est l'inverse de IP.

Lors d'une ronde, les 64 bits sont découpés en deux parties de 32 bits : G pour la partie gauche et D pour la partie droite.

Une fonction de Feistel est appliquée à D qui subit une expansion (passage de 32 à 48 bits). Ensuite, une sous-clé de ronde (48 bits) est ajoutée (XOR). Une substitution transforme ces 48 bits de façon non-linéaire. Une réduction permet de passer de 48 bits à 32 bits. Une permutation est effectuée, ce qui donne un résultat noté $F(D)$. G est ajoutée par XOR à $F(D)$ pour donner G'.

Finalement, on permute les 2 parties : D devient G et G' devient D.

La clé de ronde (figure 1.6) est générée à partir des 64 bits de la clé de chiffrement. Une première permutation avec choix ($PC-1$) est appliquée : elle sélectionne 56 bits sur les 64 bits. Ces 56 bits sont séparés en 2 parties de 28 bits. Chacune de ces parties subit une permutation circulaire d'un ou deux bits, suivant la ronde. Pour former une clé de ronde, une deuxième permutation avec choix ($PC-2$) sélectionne les 48 bits utilisés (24 dans chacune des 2 parties) dans la fonction de Feistel. On répète cette opération 16 fois pour obtenir les 16 clés de rondes.

Le Triple DES a succédé au DES lorsque la puissance de calcul des processeurs a augmenté. Le principe est d'effectuer trois DES en série avec 3 clés différentes (K_1, K_2, K_3) ce qui donne en théorie une sécurité de 168 bits.

Cependant, une attaque par rencontre au milieu la réduit à environ 112 bits. Cette attaque nécessite un texte clair et le cryptogramme correspondant. Le principe est de chiffrer le texte clair en utilisant une partie de la clé, et de déchiffrer le cryptogramme avec l'autre partie de la clé, jusqu'à obtenir une correspondance (la « rencontre au milieu »).

Dans le cas du Triple DES, on chiffre le texte clair avec toutes les paires de clés $\{K_1, K_2\}$ possibles (2^{112} chiffrements) et on déchiffre le chiffré avec toutes les clés K_3 possibles (2^{56} déchiffrements) jusqu'à obtenir une correspondance.

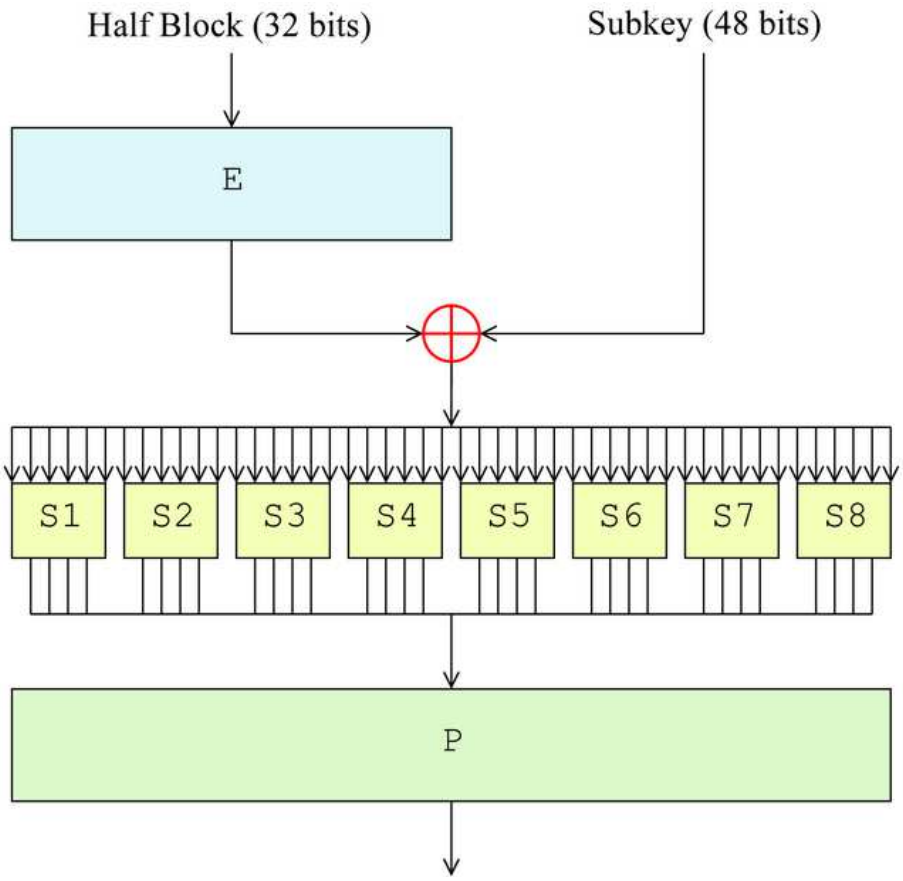

FIG. 1.5 – Fonction de Feistel du DES

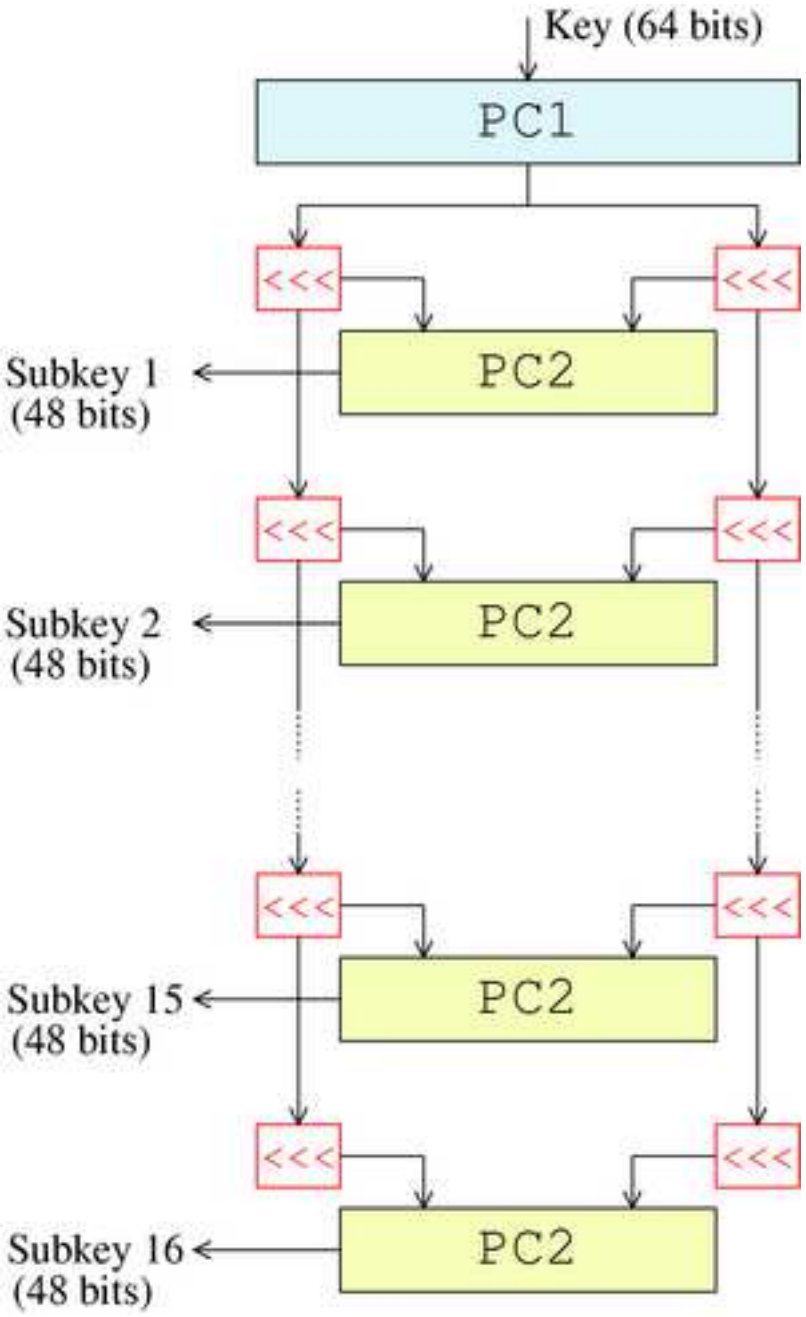

FIG. 1.6 – Génération des clés de rondes du DES

Pour ne pas utiliser des clés inutilement longues, l'algorithme est parfois utilisé avec une clé de 112 bits en utilisant la même clé pour le premier chiffrement et le dernier ($K_1 = K_3$).

De plus, comme le triple DES a été le successeur du DES, une variante existe pour remplacer le 2^{e} chiffrement par un déchiffrement. Cette variante est appelée EDE (*Encryption-Decryption-Encryption*) par opposition au mode EEE (*Encryption-Encryption-Encryption*) d'origine. Ce mode a l'avantage d'être compatible avec le DES simple lorsqu'on utilise une clé unique $K = K_1 = K_2 = K_3$.

1.2.2.2 AES

L'AES [Sta01] est le standard actuel de cryptographie symétrique par bloc du NIST. Il est plus sécurisé que le DES (clé plus longue) et plus rapide que le Triple DES en matériel et en logiciel.

C'est un réseau de substitutions et de permutations qui prend en entrée un texte clair de 128 bits et une clé de chiffrement de 128, 192 ou 256 bits.

L'algorithme d'AES est notre cas d'étude. Pour faciliter la lecture, nous avons choisi de présenter l'algorithme complet et détaillé dans la section 4.1 avec les implémentations possibles et les attaques par faute qui le concernent.

1.2.3 Cryptographie asymétrique

La cryptographie asymétrique (ou cryptographie à clé publique) [RSA78], utilise des fonctions à sens unique avec brèche secrète. Cela signifie que ces fonctions sont très difficiles à inverser, à moins de posséder une information secrète.

En pratique, la fonction à sens unique utilise une clé A et elle ne peut être inversée qu'en utilisant une clé B. Ce qui est chiffré avec A est déchiffrable avec B et réciproquement. L'une des clés est diffusée, par exemple A, qui est la clé publique, et l'autre est gardée secrète, c'est la clé privée (B).

$$\begin{aligned} Cryptogramme\_A &= Chiffrement(A, Message) \\ Message &= Déchiffrement(B, Cryptogramme\_A) \\ Cryptogramme\_B &= Chiffrement(B, Message) \\ Message &= Déchiffrement(A, Cryptogramme\_B) \end{aligned}$$

On utilise la cryptographie asymétrique pour le chiffrement, mais aussi pour l'identification et la signature. En chiffrant un message secret avec la clé publique, seul le détenteur de la clé secrète peut le déchiffrer.

Dans le cas de l'identification, on envoie un message aléatoire (appelé challenge ou défi) qui a été chiffré avec la clé publique à la personne souhaitant s'identifier. Seul le détenteur de la clé privée peut retrouver le challenge original : s'il renvoit le challenge, il sera identifié, puisqu'il aura prouvé qu'il a la clé privée correspondante. Cependant, en renvoyant un message au hasard, un attaquant a toujours la possibilité d'arriver

à s'identifier, mais avec une probabilité inversement proportionnelle au nombre de messages possibles.

$$Challenge\_A = Chiffrement(A, Challenge)$$
$$Challenge = Déchiffrement(B, Challenge\_A)$$

La signature d'un document est obtenue en chiffrant avec la clé privée le condensé (*hash*) de ce document : pour vérifier la signature, on la déchiffre avec la clé publique et on la compare au condensé du fichier reçu. Si le fichier a été modifié, le condensé diffère et il n'est pas possible de modifier la signature puisque seul le détenteur de la clé privée B peut générer des signatures déchiffrables en utilisant A.

$$Signature = Chiffrement(B, Condensé(Fichier))$$
$$Condensé\_attendu = Déchiffrement(A, Signature)$$
$$Condensé(Fichier\_reçu) = Condensé\_attendu\,?$$

Une clé de chiffrement asymétrique doit avoir des propriétés mathématiques spécifiques : il n'est pas possible de prendre une suite aléatoire de bits comme pour les clés de chiffrement symétrique. Cela explique le fait que les clés de chiffrement asymétrique sont plus longues : il est plus facile d'attaquer ces algorithmes en utilisant les propriétés mathématiques des clés, comme une factorisation dans le cas du RSA, plutôt que de réaliser une recherche exhaustive.

À cause de sa relative lenteur par rapport à la cryptographie symétrique, on utilise en pratique de la cryptographie hybride, où une clé de cryptographie symétrique est choisie au hasard pour chiffrer les données efficacement. Cette clé est ensuite chiffrée en utilisant une clé de cryptographie asymétrique A. La clé symétrique chiffrée est transmise avec les données.

Lors de la réception de la clé chiffrée et des données, le destinataire utilise B pour déchiffrer la clé symétrique choisie aléatoirement par l'envoyeur : il peut alors déchiffrer les données.

$$Cryptogramme = Chiffrement\_Symétrique(Clé\_Symétrique,\ Données)$$
$$Clé\_Chiffrée = Chiffrement\_Asymétrique(A,\ Clé\_Symétrique)$$

$$Clé\_Symétrique = Chiffrement\_Asymétrique(B,\ Clé\_Chiffrée)$$
$$Données = Chiffrement\_Symétrique(Clé\_Symétrique,\ Cryptogramme)$$

1.2.3.1 RSA

RSA [RSA78] est un algorithme asymétrique publié en 1977, publié par Rivest, Shamir et Adleman, basé sur la difficulté à factoriser efficacement les grands nombres. C'est l'algorithme de cryptographie asymétrique le plus utilisé, notamment dans le commerce électronique : HTTPS utilise TLS (anciennement appelé SSL) qui repose sur RSA.

L'algorithme de création de la paire de clés est le suivant :

1. On prend au hasard p et q, 2 grands nombres premiers, et on forme $n = pq$
2. On trouve e et d tels qu'ils sont premiers avec $(p-1)(q-1)$ et que $ed \equiv 1 \pmod{(p-1)(q-1)}$
3. Le couple (n, e) est la clé publique et le couple (n, d) est la clé privée

Le message est codé en chiffres entiers inférieurs à n. Pour chiffrer un de ces entiers M en un chiffré C, on utilise n et e pour calculer :

$$C \equiv M^e \pmod{n}$$

Le déchiffrement se fait de manière analogue, mais en utilisant n et d :

$$M \equiv C^d \pmod{n}$$

En effet,
- $ed \equiv 1 \pmod{(p-1)(q-1)}$ donc $ed = k(p-1)(q-1) + 1$
- Une généralisation du petit théorème de Fermat nous donne $M^{\Phi(n)} \equiv 1 \pmod{n}$, avec $\Phi(n)$ la fonction indicatrice d'Euler (qui correspond au nombre d'éléments de $\mathbb{Z}/n\mathbb{Z}$)
- Or, $\Phi(n) = (p-1)(q-1)$, d'où $M^{(p-1)(q-1)} \equiv 1 \pmod{n}$
- D'où $C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^{k(p-1)(q-1)+1} \equiv M \pmod{n}$

La sécurité de RSA repose sur deux conjectures : il faut factoriser n pour casser RSA et il n'est pas possible de factoriser un nombre en temps polynomial. Cela implique qu'en choissant n suffisamment grand, le RSA est considéré sûr, et cela fait 25 ans qu'il est cryptanalysé, sans succès.

RSA Laboratories organise un concours permanent qui consiste à décrypter des messages chiffrés avec des tailles de clé de plus en plus longues. Actuellement, le record de factorisation porte sur un chiffre de 663 bits et date de 2005. La taille de clé minimum utilisée en 2007 est de 1024 bits, mais le NIST (États-Unis), la DCSSI (France) et le BSI (Allemagne) recommandent d'utiliser des clés de 2048 bits à partir de 2011.

1.2.3.2 Courbes elliptiques

La cryptographie basée sur des courbes elliptiques utilise des courbes vérifiant l'équation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

que l'on utilise sous la forme simplifiée

$$y^2 = x^3 + ax + b$$

sur les corps de Galois $GF(n)$. L'usage des courbes elliptiques en cryptographie a été proposé, de manière indépendante, par Neal Koblitz et Victor Miller en 1985. Les algorithmes basés sur les courbes elliptiques utilisent des clés plus courtes que le

RSA. Cela permet de combiner des clés asymétriques de 384 bits à des algorithmes symétriques de 256 bits, alors que le RSA nécessiterait des clés de 15000 bits à sécurité comparable.

Cependant, les opérations de chiffrement et de déchiffrement nécessitent plus d'opérations à effectuer et à coordonner. Même si la cryptographie sur courbes elliptiques est très prometteuse, elle est encore mal connue parce que les standards sont relativement jeunes et un grand nombre de brevets freine son développement.

Une courbe elliptique sur un corps fini, notée $E(a, b, n)$, est de la forme $y^2 = (x^3+ax+b)$ en choisissant a et b sur ce corps fini (défini par un polynôme irréductible sur $\mathbb{Z}/n\mathbb{Z}$) et où les opérations d'addition et de multiplication sont celles du corps fini sur lequel on se place. Les couples (x, y) qui vérifient cette équation sont les coordonnées de points qui forment un groupe abélien lorqu'ils sont munis d'une opération d'addition.

Cela signifie que pour tous points P, Q, R de la courbe :
- $P + Q = Q + P$
- $(P + Q) + R = P + (Q + R)$
- il existe un élément neutre O tel que $P + O = O + P = P$
- il existe $-P$ tel que $(-P) + P = P + (-P) = O$

Cela entraine l'existence de la multiplication d'un point P par un scalaire k : $kP = P + (k - 1)P$, avec $1P = P$ et $-kP = k(-P)$.

L'ensemble des kP défini alors un sous-groupe cyclique de la même manière que l'exponentiation modulaire dans le cas du RSA. On peut alors définir de manière analogue un algorithme de chiffrement qui utilise ce groupe : p, q, n, e, d sont choisis de la même manière.

Le sous-groupe kP a un cardinal qui divise le cardinal de l'ensemble des points : P (avec $P \neq O$) est générateur d'un groupe des points de la courbe de cardinal p, q ou n.

Le chiffrement et le déchiffrement se font par multiplication scalaire : $C = eM$ et $M = dC$.

La cryptographie sur courbes elliptiques, tout comme les autres algorithmes de chiffrement asymétrique, peut aussi être utilisée pour l'établissement d'un canal de communication sécurisé.

Supposons que 2 personnes (A et B) souhaitent établir un secret partagé sur un canal qui n'est pas sécurisé. Il choisissent publiquement :
- $E(a, b, p)$, une courbe elliptique
- P, un point de cette courbe, de préférence générateur de la courbe

Secrètement, A choisit un entier d_A, et B un entier d_B. A envoie le point d_AP, et B envoie d_BP. Chacun calcule $d_A(d_BP) = (d_Ad_B)P$ qui est un point de la courbe, et constitue leur clef secrète commune.

Un espion connaît $E(a, b, p), P, d_AP, d_BP$. Pour pouvoir calculer d_Ad_BP, il faut pouvoir calculer d_A connaissant P et d_AP, c'est-à-dire résoudre le problème du lo-

garithme discret sur une courbe elliptique. Or, actuellement, si les nombres sont suffisamment grands, on ne connaît pas de méthode efficace pour résoudre ce problème en un temps raisonnable.

1.2.4 Attaques sur les circuits et contre-mesures

Les systèmes microélectroniques sont la cible des attaques qui exploitent les propriétés physiques du système, comme l'analyse de courant électrique ou électromagnétique. Ils peuvent aussi être attaqués en utilisant des attaques issues de l'informatique comme le dépassement de tampon (*buffer overflow*).

Les attaques présentées ici nécessitent des moyens et des connaissances variables. Les attaquants peuvent être représentés par trois archétypes :

- le particulier, qui a peu de moyens financiers et utilise les connaissances disponible publiquement (ex : hacker curieux)
- les institutions académiques, qui ont un peu plus de moyens et des connaissances plus pointues (ex : laboratoire de recherche)
- les organisations et états, qui disposent de moyens considérables mais pas des connaissances, ce qui les amène à financer les deux premiers

Il faut souligner que ces attaques s'appliquent aux implémentations et non aux algorithmes eux-mêmes. En effet, il est évident, par exemple, qu'une attaque en courant (voir paragraphe suivant) ne s'applique qu'aux systèmes électriques : ce n'est pas l'algorithme qui est vulnérable mais le système qui l'implémente. Autrement dit, ces attaques ne remettent pas en cause la sûreté théorique des algorithmes.

1.2.4.1 Attaques par canaux cachés

Les attaques par canaux cachés se basent sur l'observation des conséquences physiques du déroulement du circuit, c'est-à-dire sur les effets de bords de son fonctionnement. Ce sont des fuites d'informations qui permettent de retrouver des informations secrètes contenues dans un circuit.

Elles sont inspirées des méthodes biologiques de quantification, comme par exemple la calorimétrie, qui estime le nombre de bactéries en mesurant la chaleur qu'elles produisent. L'analogie avec les circuits a donné lieu à l'analyse de courant (« nourriture » consommée) et l'analyse du rayonnement électromagnétique (énergie rejetée).

Les attaques par canaux cachés ont souvent 2 variantes : celle où la fuite d'information donne directement une information secrète (Simple ...) et celle qui nécessite plusieurs mesures pour supprimer le bruit mesuré (Differential ...). Dans le deuxième cas, on s'appuye sur le fait que le bruit est aléatoire, ce qui signifie qu'il est constant en moyenne.

SPA/DPA Les micro-circuits utilisés dans les cartes à puce sont élaborés en technologie CMOS. Le transistor MOS complémentaire (CMOS) est composé d'un tran-

sistor MOS de type N (NMOS) et d'un transistor MOS de type P (PMOS). La figure 1.7 donne le schéma de base d'un inverseur (porte élémentaire) CMOS qui est l'élément de base de toutes les fonctions logiques.

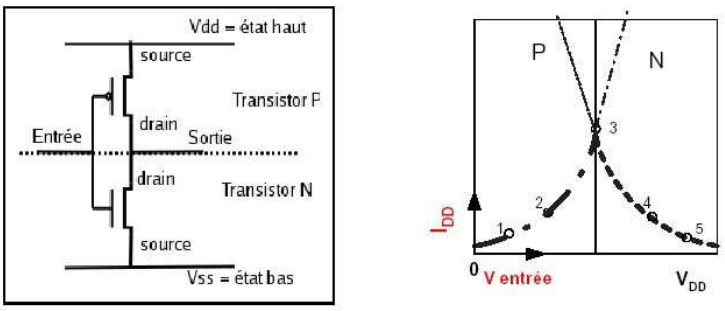

FIG. 1.7 – Schéma et caractéristique d'une porte CMOS

Lorsque la tension de sortie $V_0 = VDD$ ou VSS l'un des deux transistors est à l'état bloqué et le courant traversant les transistor entre VDD et VSS est alors négligeable. Or les deux transistors ont un fonctionnement antagoniste : quand l'un est passant l'autre est bloqué, et vice-versa. Cependant, lors du basculement des transistors d'un état bloqué vers un état passant, il va exister un très court instant pendant lequel deux transistors vont se trouver en état de court-circuit. Cet état va se traduire localement par une forte augmentation de la consommation de courant. Toute l'analyse des fuites en consommation de courant repose sur ce phénomène de basculement de porte qui est très dépendant de la technologie utilisée, ainsi que des données qui transitent à travers ces portes.

La SPA (*Simple Power Analysis*, analyse simple de courant) [KJJ99] consiste à retrouver un comportement en observant le courant consommé par le système. Si ce comportement change suivant les données secrètes manipulées, il est possible d'obtenir des informations importantes sur les secrets embarqués. Une implémentation de RSA sur carte-à-puce peut ainsi faire apparaitre directement les données de la clé. Sur la figure 1.8, un motif de la courbe de consommation correspond à un 0 (*Square*, correspondant à une mise au carré) et un autre à un 1 (*Square-and-Multiply*, correspondant à une mise au carré suivi d'une multiplication). Il est aussi possible d'obtenir des indices sur les instructions exécutées en interne, ou de détecter l'activité d'une partie du circuit.

La DPA (*Differential Power Analysis*, analyse différentielle de courant) [KJJ99] est une attaque basée sur l'analyse statistique des variations de consommation d'un système. Le principe est de retrouver la consommation d'un nombre réduit de bits (usuellement un seul) à travers une moyenne de la consommation du circuit pour plusieurs textes clairs.

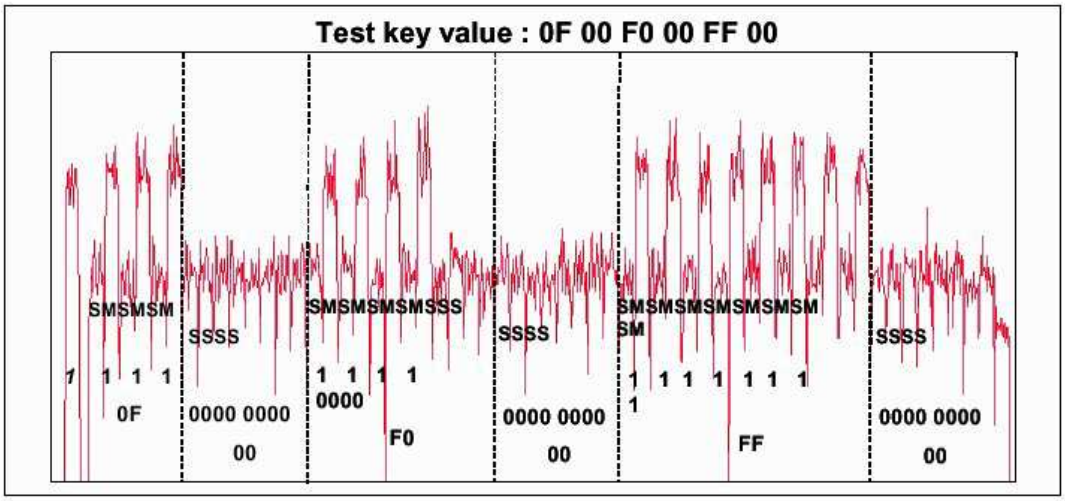

FIG. 1.8 – SPA sur une implémentation de RSA

Pour effectuer une DPA, il faut commencer par trouver une fonction de sélection qui prédit un bit D à partir d'un texte d'entrée et d'une valeur de clé partielle. Par exemple, cela peut être un bit de la sortie du premier SubBytes de l'AES, calculé à partir du message d'entrée et des huits premiers bits de la clé.

Ensuite, on effectue les mesures des consommations sur le circuit pour un ensemble de textes choisis.

Pour chacune des 2^8 clés partielles, on range ces textes en deux groupes, suivant si $D = 0$ ou si $D = 1$ avec la clé choisie. Enfin, on effectue la moyenne des consommations de ces groupes : si une différence apparait, c'est que la clé partielle choisie est la bonne, puisque la différence de consommation du bit D correspond.

SEMA/DEMA La SEMA et la DEMA (*Simple/Differential ElectroMagnetic Analysis*, analyse simple/différentielle électromagnétique) [QS01] sont similaires à la SPA et la DPA, mais elles se basent sur le rayonnement électromagnétique.

Comme un courant électrique qui passe dans un conducteur produit nécessairement un rayonnement, il est possible d'utiliser cette fuite d'énergie pour réaliser une attaque par canal caché.

Le principe consiste à corréler des traces de rayonnement électromagnétique et les bits du secret manipulé dans la mesure où ce rayonnement varie en fonction des données secrètes manipulées.

Le principal avantage de la DEMA sur la DPA est qu'elle peut être réalisée localement en mesurant le rayonnement d'une partie du circuit seulement et qu'elle ne nécessite pas de contact avec le circuit. Cependant, cette technique est plus complexe en pratique.

Attaques en temps L'efficacité des attaques en temps [Koc96] provient de la différence de temps de calcul dépendant des données, par exemple lorsque les operandes sont courtes ou quand les algorithmes sont optimisés pour les cas les plus fréquents des données.

Par exemple, une vérification naïve d'un mot de passe est faite caractère par caractère. Ainsi, si la première lettre du mot proposé est correcte, le refus prendra plus de temps que si toutes les lettres sont fausses. On peut alors récupérer le mot de passe en testant toutes les lettres pour chacun des caractères. Cela transforme la recherche exhaustive exponentielle en recherche linéaire.

Autres canaux cachés possibles D'autres canaux cachés peuvent être envisagés, même s'ils semblent moins prometteurs.

L'analyse en température est basée sur la chaleur émise par le système. L'approche est similaire à la DPA, mais en utilisant la dissipation en chaleur.

La cryptanalyse acoustique consiste à analyser les sons émis par le système effectuant des opérations cryptographiques comme un cryptoprocesseur. Ce type d'attaque date de l'époque des machines cryptographiques, après la Seconde Guerre mondiale, avec l'analyse des sons émis par les touches ou les rotors.

L'approche la plus probable est maintenant l'utilisation conjointe de plusieurs fuites d'informations pour réaliser des attaques mixtes ou combinées. Par exemple, l'analyse de la consommation couplée à l'analyse du rejet électromagnétique peut ouvrir de nouvelles possibilités d'attaques.

Contremesures Dans un premier temps, le concepteur de circuit va essayer de rendre le fonctionnement du circuit régulier, par exemple en effectuant toujours tous les calculs possibles puis en choisissant le résultat (plutôt que de n'effectuer que le calcul nécessaire) ou en compensant un comportement par un autre : lorsqu'un fil passe à 1, un autre passe à 0 et l'état n'est valide que lorsque les deux fils sont complémentaires (logique double-rail).

Lorsque ce n'est pas possible, les contremesures usuelles contre les attaques par canaux cachés se basent principalement sur l'ajout de protections destinées à rendre plus difficile l'exploitation des mesures physiques.

Ajouter du bruit aléatoire permet de brouiller les mesures de grandeurs physiques, par exemple en effectuant des calculs inutiles pour brouiller la consommation de courant et le rayonnement électromagnétique.

Le circuit peut aussi être muni d'un système ajoutant aléatoirement des cycles de calcul pour que les calculs utiles n'aient pas toujours lieu au même moment. L'ajout d'une horloge instable permet de décaler légèrement et progressivement les fronts d'horloge pour désynchroniser les courbes.

1.2.4.2 Attaques invasives

Les attaques invasives visent à dégrader physiquement le circuit pour obtenir des informations. Par exemple, un attaquant peut essayer de désactiver un système de protection en coupant des connexions internes du circuit.

Ce type d'attaque nécessite du matériel perfectionné et très couteux et une mauvaise manipulation peut détruire le circuit sans fournir les informations attendues.

Par exemple, une attaque par sondage (*probing attack*) consiste à retirer des couches de métal du circuit pour pouvoir se connecter aux lignes des bus de données du circuit et espionner les données qui transitent, ou injecter des signaux parasites. Avec un accès en lecture, il est possible de chercher à retrouver des valeurs sensibles avant, après ou pendant un calcul. Une protection possible est le chiffrement à la volée des données sur le bus et dans la mémoire [ELB06].

Les techniques issues des attaques invasives ont depuis été utilisées pour faciliter d'autres types d'attaques comme les attaques par faute : il est plus facile de perturber un circuit en enlevant ou en désactivant ses protections.

1.2.4.3 Attaques par faute

Une attaque par faute sur un système consiste à perturber physiquement une ou plusieurs parties de celui-ci en vue de le corrompre et d'en exploiter les comportements erronés. Le principe est d'exécuter des calculs inter-dépendants à une faute près, permettant d'extraire par cryptanalyse des données protégées. Ainsi, injecter des fautes dans un circuit peut aider à déjouer ses protections sécuritaires, comme le montrent les attaques par DFA (*Differential Fault Analysis*, analyse différentielle de faute).

Les algorithmes de cryptographie doivent donc être sûrs contre la cryptanalyse mathématique, mais aussi résistants contre les attaques par faute, notamment lorsqu'ils sont implémentés en matériel. Avec plusieurs de ces chiffrés erronés et les chiffrés corrects, l'attaquant déduit des informations sur la clé secrète. Plusieurs publications présentent des attaques par faute sur différents algorithmes, en utilisant les différences des sorties : RSA [BDL97], DES [BS97] et AES [Gir04, DLV03, PQ03, MSS06].

Les attaques en Safe Error [YJ00, BS03] exploitent le fait que le résultat en sortie change ou non lors de l'injection de fautes de collage. Si un bit collé à 0 ne change pas le résultat du circuit, c'est que ce bit était naturellement à 0 dans son exécution sans fautes.

La DBA (*Differential Behavioural Analysis*, analyse différentielle de comportement) [RM07] est une analyse similaire à la DPA mais au lieu de se baser sur des mesures de consommation, on utilise les changements en sortie lorsqu'on injecte une faute de collage (c'est-à-dire lorsqu'on force la valeur d'un bit à une valeur définie).

La fonction de sélection dépend du texte d'entrée, d'une faute de collage et d'une clé partielle. Pour chacun des textes d'entrée, l'injection de la faute de collage

change ou non la sortie, suivant si le collage modifie la valeur ou non. On note si la sortie a changé ou non dans une liste, qui est la trace de l'exécution. Cette trace du comportement en faute est précalculée pour chaque clé partielle.

Ensuite, on réalise le chiffrement de chacun des textes d'entrée en injectant physiquement le collage et on regarde si le résultat est modifié ou pas en sortie. La trace obtenue doit correspondre à la trace pré-calculée utilisant la clé partielle correspondante.

Par soucis de coût et de flexibilité, certains types de systèmes microélectroniques possèdent une partie logicielle. Cela permet la mise à jour de données et l'ajout de fonctionnalités sur un socle physique défini. Par exemple, certains systèmes sont basés sur des circuits utilisant un jeu d'instructions et où les fonctionnalités sont programmées en logiciel. Une application est divisée en deux parties : les données qui sont manipulées par le programme et le code, qui est la suite d'instructions exécutée sur ces données. C'est le schéma classique de Von Neumann [VN82], qui est encore très utilisé pour les applications embarquées, même si on considère pourtant la séparation du code et des données comme une bonne pratique de programmation.

La première attaque possible se base sur une modification des données. Changer les données manipulées peut changer complètement le comportement du système.

Certaines attaques par faute forcent la réinitialisation de la mémoire à un état donné comme 00 ou FF pour un octet [BECN+04]. Ces méthodes permettent par exemple la réinitialisation de compteurs, la réduction du nombre de rondes d'un calcul cryptographique [CT05], mais aussi les attaques cryptographiques utilisant des textes clairs connus. Le chiffrement des données est une bonne protection, mais il existe d'autres solutions moins onéreuses. Par exemple, on peut utiliser un codage d'information avec bit de parité inversé : le dernier bit est l'inverse de la parité des 7 autres. Dans ce cas, 00 et FF sont des valeurs invalides.

Avec un accès en lecture et en écriture, il devient possible de copier ou déplacer certaines données dans la mémoire. Même si les données sont chiffrées, il est alors possible d'inverser 2 variables ou de cloner une valeur, à la condition de connaitre leur localisation. Une protection robuste contre ce type d'attaque est le chiffrement des données en utilisant une clé secrète dépendante de l'adresse d'écriture [ELB06] : si l'adresse change, la donnée chiffrée doit aussi être modifiée pour rester valide.

Le code stocké en mémoire peut aussi être la cible d'attaques : des instructions sont modifiées ou ignorées. Le changement d'une instruction peut modifier les données mémorisées, par exemple lorsqu'un compteur est décrémenté plutôt qu'incrémenté. Ce cas se rapproche de la modification des données.

D'autre part, le flot d'exécution du programme peut être altéré. Si un test est modifié, un ensemble d'instructions sera exécuté malgré le fait qu'il n'est pas adapté à l'état actuel du système. De la même manière, si le pointeur d'instruction courante (*stack pointer*) est modifié, des instructions pourront être répétées ou ignorées.

Plus généralement, la modification d'une valeur de saut peut dérouter complètement le système de son exécution correcte. C'est notamment le cas dans les attaques par dépassement de tampon (*buffer overflow*) [Ale96] où le flot d'exécution est dérouté vers l'espace de données, permettant ainsi l'exécution de code défini par l'attaquant.

Cependant, la mise en œuvre d'une attaque par faute se heurte à plusieurs problèmes. Les hypothèses d'injection sont plus ou moins réalisables en pratique, notamment en ce qui concerne le lieu, le moment et le type d'injection de faute. Les perturbations physiques injectées [SA02] [BECN+04] sont souvent difficiles à analyser et difficilement reproductibles [MRL+06].

De plus, il faut appliquer une perturbation physique du circuit qui ne soit pas détectée par les systèmes de détections existants. Les contremesures usuelles qui permettent la détection d'attaque utilisent des capteurs physiques et/ou de la redondance d'information. Par exemple, des capteurs de lumière peuvent protéger un circuit contre les attaques par faute par laser, ou les adresses et/ou les données peuvent être codées sur un nombre plus important de bits pour détecter les corruptions en mémoire ou même les corriger en utilisant des codes correcteurs.

Actuellement, ces attaques se développent rapidement parce que les perturbations dépendent d'un grand nombre de paramètres physiques du circuit et sont donc difficilement prévisibles. Ce travail de thèse s'inscrit dans la volonté de prédire les effets de ces fautes et d'évaluer les protections des circuits.

1.3 Outils d'injection de fautes

Pour concevoir des circuits tolérants aux fautes, il est nécessaire de prendre en compte le comportement du circuit lors de l'occurence de fautes. La robustesse contre les attaques par faute doit être évaluée pour garantir la sécurité et la tolérance aux fautes.

La méthode la plus immédiate est d'injecter des fautes dans le circuit et d'observer son comportement. L'injection de fautes peut être réalisée sur un circuit physique ou en simulation.

Dans le cas d'un circuit physique, l'injection peut être faite de manière physique avec un laser [SA02], une injection d'ions lourds [KLD+94, MRCV99, CNV+00] ou des pics de tension sur l'alimentation [MHGT92]. L'autre possibilité est d'utiliser les fonctionnalités intégrées de debug pour accéder à l'état interne du circuit.

En simulation, le circuit peut être simulé sans modification et les fautes sont injectées en utilisant le simulateur [CI92] ou la description du circuit peut être modifiée pour tenir compte du modèle de faute choisi [STB97].

C'est une technique qui permet d'estimer le comportement du circuit avant même de l'avoir réalisé physiquement. Ainsi, il est possible de tester plusieurs contremesures pour implémenter physiquement seulement celle qui donne les meilleurs résultats de protection.

Les outils et méthodes d'injection de fautes en simulation existent depuis une vingtaine d'année. Nous présentons brièvement et de manière non-exhaustive certaines caractéristiques d'outils existants en insistant sur les modèles de fautes associés. Nous avons choisi d'élargir notre état de l'art aux outils d'injection de fautes en général (injection sur matériel, mais aussi logiciel, OS, réseau) pour avoir une vision globale de la problématique et des solutions possibles.

1.3.1 Injection physique

Les outils d'injection physique sont ceux qui attaquent directement le système au niveau matériel, en perturbant physiquement le circuit, dans le but de valider physiquement le système. Ces techniques ne sont pas intrusives dans la mesure où le comportement original du circuit n'est pas modifié.

Ce type d'injection peut coûter cher car certaines attaques nécessitent un équipement spécifique et couteux, en plus du coût de construction du système. En effet, l'injection physique se fait en aval de la conception. C'est pour cette raison que l'injection physique de faute ne peut être utilisée pour une évaluation de la sécurité des circuits avant qu'il ne soit effectivement fabriqué.

L'injection de fautes physiques peut être avec ou sans contact.

Lorsque l'injection de faute utilise un contact direct avec le matériel, il peut appliquer des changements de voltage ou de courant. Par exemple, c'est le cas des méthodes de forçage et d'insertion au niveau broche, ainsi que les attaques par sondage.

S'il n'y a pas de contact physique direct, l'injection est sans contact, et une source externe produit un phénomène physique qui induit des fautes à l'intérieur du circuit. C'est notamment le cas de l'injection d'ions lourds ou des fautes provoquées par laser.

Les injections de fautes matérielles ciblent des exemples réels de circuit après fabrication. Le circuit est soumis à des perturbations qui engendre des erreurs et le comportement obtenu est analysé. Suivant les techniques d'injection de fautes, certains types de fautes sont possibles ou non. La mise en œuvre des expériences nécessite du temps pour préparer le test du circuit. Cepdendant, ce temps est compensé par la vitesse de réalisation des injections, ce qui permet de lancer des applications plus complexes et par conséquent d'obtenir une analyse comportementale plus réaliste du circuit en présence de fautes.

FIST, MESSALINE et RIFLE sont trois outils d'injections de fautes physiques. Nous avons choisi FIST pour la proximité de son mode opératoire par rapport aux fautes réelles observées dans l'aéronautique. MESSALINE propose une injection orientée test plus proche du mode opératoire des attaques par faute, où le circuit peut être isolé et manipulé. RIFLE a apporté des améliorations concernant les processeurs plus complexes et l'analyse des résultats.

FIST [GKT89] (Fault Injection system for Study of Transient fault effects) est un outil d'injection de fautes par radiation d'ions lourds (Californium 252), développé à l'université Chalmers de Göteborg (Suède) en 1989.

Le système est introduit dans une chambre sous vide dans laquelle est effectuée la radiation, en vue d'étudier sa sûreté de fonctionnement.

Une étude de cas a été réalisée sur un Motorola MC6809E renforcé par des mécanismes de détection de fautes. FIST est conçu pour détecter les changements de bits (bit-flip), mais aucun modèle de faute au niveau bit n'est décrit.

MESSALINE [AAA+90] a été développé au LAAS-CNRS de Toulouse en 1989, dans le but de tester les mécanismes de tolérance aux fautes.

MESSALINE injecte des fautes au niveau interconnexions sur un prototype du système. La faute est injectée directement sur l'une des interconnexions du circuit cible, et il est possible de contrôler plusieurs paramètres :

- l'endroit de la faute dans le système
- le type de faute (stuck-at-0, stuck-at-1, short, open,... etc.)
- les caractéristiques temporelles (temporaire, permanente, intermittente)
- le moment d'application de la faute
- la durée d'application de la faute

MESSALINE et le système sont reliés à travers quatres modules en utilisant, par exemple, une interface parallèle. Le module d'injection peut cibler simultanément jusqu'à 32 points d'injection sur les interconnexions. Le module d'activation assure l'initialisation du système cible. Le module de collection affiche le contenu des éléments observés, comme les signaux d'erreurs ou les sorties du système. Le module de gestion conduit la génération automatique des séquences de test, contrôle le temps d'exécution et archive les résultats pour des analyses ultérieures.

RIFLE [MRMS94] est un outil d'injection de fautes au niveau interconnexion développé à l'université de Coimbra au Portugal en 1994.

L'architecture de RIFLE nécessite que le système testé soit connecté à l'ordinateur qui exécute RIFLE à travers des cartes d'interfaces et un module d'adaptation, qui sont spécifiques au système testé. RIFLE est capable de distinguer les fautes n'ayant pas provoqué d'erreurs des fautes dont les erreurs ont été ensuite écrasées par le fonctionnement du circuit.

Par rapport à MESSALINE, RIFLE est plus ciblé vers le test de microprocesseurs plus complexes et qui utilisent des techniques comme le prefetching, le pipelining, etc.

Une faute est définie par son type, sa date de début, sa durée et les interconnexions sur lesquelles elle s'applique. Les types de fautes possibles sont :

- collage à 0, 1 ou à une valeur externe
- inversion
- contact avec la interconnexion suivante
- contact avec la interconnexion précédente
- circuit ouvert

Le tableau 1.9 fournit une synthèse de ces outils. L'injection de fautes physiques est une méthode non-intrusive : aucune modification n'est nécessaire pour l'appliquer à un circuit. C'est une méthode rapide puisque l'expérience peut être exécutée à la même vitesse que le circuit lui-même. L'environnement d'injection est comparable à celui dans lequel le circuit sera placé par la suite, assurant une grande représentativité aux fautes injectées.

Néanmoins, ces techniques exigent d'avoir accès au circuit, ce qui n'est pas évident dans le cas de systèmes intégrés. Le circuit sur lequel les fautes sont injectées peut présenter un grand risque d'endommagement qui dépend du type d'injection physique, même si on s'assure en pratique de limiter au maximum les dégâts. Le matériel nécessaire à la réalisation des injections peut être couteux et difficile à obtenir. De plus, l'observabilité et la contrôlabilité sont souvent limitées.

Nom	**Année**	**Type d'injection**	**Modèle**
FIST	1989	Radiation d'ions lourds	Bit flips
MESSALINE	1989	Injection sur les interconnexions	Valeur choisie injectée n'importe où et quand dans le système
RIFLE	1994	Injection sur les interconnexions	Stuck-at, inversion, contact avec autre interconnexion, circuit ouvert

FIG. 1.9 – Synthèse des outils d'injection physique

1.3.2 Injection logicielle

Les outils d'injection logicielle injectent des fautes pendant l'exécution normale du système, en utilisant des mécanismes internes au système, comme le logiciel exécuté sur le système, les mécanismes intégrés de debug, de scan-chain ou d'accès direct à la mémoire (JTAG). En général, la méthode consiste à changer l'état du système dans la partie mémorisante (registres, RAM) ou à modifier le résultat d'un calcul (corruption d'un résultat, modification d'un message interne).

L'injection peut être faite à la compilation : le système est modifié pour émuler l'effet de la faute. On parle alors de mutation du système. Cela a l'avantage d'intégrer facilement la problématique des fautes dans le flot de conception existant puisqu'elles sont prises en compte par le système lui-même. Néanmoins, les fautes sont statiques puisqu'elles sont déterminées à la compilation.

L'autre possibilité est d'émuler la faute lors de l'occurence d'un évènement. La technique la plus simple utilise une temporisation pour injecter la faute à un moment choisi à partir du début du déroulement du système. En utilisant les mécanismes d'exception logicielle ou d'interruption matérielle, la faute peut être émulée lorsqu'une condition spécifique est remplie, par exemple lorsqu'une instruction particulière est exécutée.

L'injection logicielle est attractive car elle permet de réduire le coût et de fournir une alternative de contrôle plus facile comparée aux techniques d'injection physiques qui exigent un équipement matériel spécial et des interfaces spécifiques au système cible. Par ailleurs, ces injections visent souvent l'application et éventuellement le système d'exploitation qui s'exécute sur le système cible, ce qui est difficile à effectuer en utilisant des techniques d'injections basées sur le matériel.

FERRARI est l'un des premiers outils d'injection de fautes émulant les effets des fautes physiques dans les programmes par des moyens logiciels. Xception a affiné le modèle de fautes tout en ciblant le processeur plutôt que les programmes. EXFI a proposé des travaux qui permettent d'utiliser les fonctions du processeur au lieu de celles du système d'exploitation, ce qui permet d'utiliser des processeurs utilisés dans l'informatique embarqué. MAFALDA a permis de prendre en compte l'architecture des micro-noyaux et de tester à la fois les applications et le système d'exploitation face aux fautes.

FERRARI [KKA95] (Fault and ERRor Automatic Real-time Injector) a été créé à l'université d'Austin (Texas) par Kanawati, Kanawati et Abraham en 1992, pour évaluer la sûreté de fonctionnement de systèmes complexes en émulant des fautes matérielles par logiciel.

Il cible un programme sur la même machine que l'injecteur et utilise des déroutements (« trap ») matériels et logiciels pour injecter les fautes dans l'image mémoire au moment voulu.

Les fautes peuvent être temporaires ou permanentes et peuvent être injectées en mémoire avant le début du programme, après un certains nombre d'apparitions d'une adresse du segment de code ou au bout d'un certain temps. Il est possible de choisir la durée de la faute. Toutes les fautes sont émulées en logiciel.

Les fautes temporaires ont été déterminées à partir de modèles de fautes matérielles. Ces fautes temporaires peuvent être une erreur sur le compteur impliquant l'exécution d'une autre instruction, une erreur impliquant l'exécution de 2 instructions, une erreur lors de la recherche d'une opérande de donnée, une erreur de stockage d'une opérande, une erreur d'interprétation du code d'opération, une erreur

de chargement ou du stockage d'une opérande ou une erreur dans les drapeaux de condition.

Les fautes permanentes sont de 3 types : faute sur la ligne d'adresse (déplacement du compteur du programme), sur la ligne de donnée (changement de l'instruction) et faute sur le code de condition.

FERRARI peut aussi injecter des fautes dans le noyau du système d'exploitation en utilisant un processus privilégié.

Xception [CMS98] a été créé à l'université de Coimbra au Portugal en 1995, pour proposer un outil d'injection de fautes utilisant les fonctionnalités de debug et de surveillance des processeurs.

L'architecture d'Xception comporte un ordinateur hôte et un système cible. Xception est constitué de 3 modules : le module noyau (cible), le module d'installation des fautes (cible) et le module de gestion de test (hôte). Le module noyau est lié statiquement au noyau du système d'exploitation et contient les gestionnaires d'exceptions. Ce nouveau noyau doit être utilisé à la place du noyau normal pour réaliser les expérimentations. Le module d'installation des fautes est une bibliothèque qui reçoit les paramètres de la machine hôte et les passe au noyau. Le module de gestion de test est localisé sur le système hôte et permet la définition des fautes injectées, le contrôle de l'expérimentation et la collection des résultats.

Les fautes considérées sont toutes temporaires, car forcer la valeur d'une variable pendant toute l'exécution est quelque chose de difficile. Elles peuvent avoir lieu dans la plupart des parties du processeur : Instruction Execution Control unit, Integer Unit, Floating Point Unit, Memory Management Unit, Internal Data Bus, Internal Adress Bus, General Purpose Registers et Condition Code Register. Ces parties sont celles du modèle générique d'Xception, défini comme une interface commune aux processeurs utilisables avec Xception. Les fautes peuvent être injectées en utilisant des triggers : opcode fetch, operand load, operand store, temps depuis le début ou une combinaison de ceux-ci. Elles peuvent être des stuck-at-0, des stuck-at-1, des bit-flips avec un masque de bits possibles (32 bits) ou des partages entre deux bits (« bridging »). Par exemple, on peut spécifiér une faute sur 3 bit-flips avec un masque 0x000000FF pour que les erreurs ne soient que sur les bits de poids faibles.

EXFI [BPRR98] (EXception based Fault Injector) a été développé en 1998 à l'école polytechnique de Turin. Selon ses développeurs, ses principaux avantages sont qu'il est économique, portable et efficace.

EXFI est basé sur le *Trace Exception Mode* que l'on trouve dans les microprocesseurs. Il utilise exclusivement des exceptions pour injecter des fautes.

Par rapport à FERRARI, EXFI cible les microprocesseurs embarqués, plutôt que ceux de station de travail, ce qui l'amène à utiliser les capacités du microprocesseur et non celles du système d'exploitation. De plus, EXFI ne modifie par le code cible, ce qui le rend moins intrusif.

Les fonctionnalités de debug spécifiques à un processeur ne sont pas utilisées, à

l'inverse d'Xception sur le processeur PowerPC. EXFI s'appuie sur le mécanismes de détection d'erreur (EDM du processeur) pour détecter un comportement erroné.

EXFI injecte des bit-flips temporaires dans la mémoire (code & données) et dans les registres utilisateurs. L'approche peut être étendue aux collages, couplages, multiple bit-flips temporels et spatiaux, etc.

Le moment de la faute est exprimé en nombre d'instructions depuis le début de l'exécution de l'application.

MAFALDA [AFRS03] a été développé au LAAS-CNRS à Toulouse en 1999, dans le but de tester la robustesse et d'aider la conception des micro-noyaux. Il a été adapté pour les systèmes temps-réels [RAA02].

Pour injecter des fautes, MAFALDA intercepte les communications (appels de fonction,...) entre les processus et le micro-noyau pour pouvoir les modifier. L'injection est pilotée par une machine hôte qui récupère les rapports (logs) des processus, de l'injecteur et du détecteur d'erreur.

MAFALDA injecte des bit-flips dans les paramètres utilisés dans l'utilisation des primitives du micro-noyau ou dans l'image mémoire du micro-noyau (segment de code ou de données), s'appuyant sur plusieurs études qui prouvent que ce modèle représente correctement les fautes physiques.

Comme elle ne nécessite pas de matériel spécifique, l'injection logicielle a connu un gros succès au milieu des années 90 : plus d'une dizaine d'outils entre 1992 et 1999 dont les principaux sont résumés dans le tableau 1.10.

L'injection logicielle émule les fautes physiques en modifiant l'état du système cible. Cette technique permet d'injecter des fautes dans des applications et des systèmes d'exploitation, ce qui est plus difficile à concevoir avec des injections de fautes par matériel. Lors des campagnes d'injection, le système s'exécute à la même vitesse (ou presque) qu'en fonctionnement, ce qui permet de réaliser un grand nombre d'injections et de tester plusieurs modèles de fautes. De plus, l'injection logicielle ne nécessite aucun matériel dédié puisqu'elle réutilise les capacités fournies par le système à tester.

Cependant, les injections ne sont possibles que dans les endroits accessibles au logiciel. L'observabilité et la contrôlabilité sont donc parfois limitées. Les modifications apportées au système peuvent conduire à des résultats erronés, comme des interférences au niveau du temps d'exécution, notamment lorsque le processus d'injection est ajouté au système.

1.3.3 Injection en simulation

Lorsque le système cible est simulé, l'injection de fautes est encore plus simple, puisque l'état du circuit est complètement manipulable par le simulateur. La simulation permet l'analyse d'un système à plusieurs niveaux, depuis la description fonctionnelle jusqu'au comportement physique. Elle utilise des langages comme VHDL

Nom	Année	Cible d'injection	Type de fautes
FERRARI	1992	image mémoire d'un programme	niveau « assembleur » (instructions, données)
Xception	1995	processeur d'une machine réseau	CPU, registres, bus
EXFI	1998	carte embarquée basée sur microprocesseur	bit-flip mémoire et registres
MAFALDA	1999	micro-noyaux	bit-flips sur les paramètres d'appel des primitives et sur l'image mémoire du micro-noyau

FIG. 1.10 – Synthèse des outils d'injection logicielle

ou Verilog, parce qu'ils permettent de modéliser le circuit aussi bien au niveau comportemental (algorithmes) qu'au niveau structurel (portes logiques).

À un niveau plus élevé, il est possible d'effectuer des tests par co-simulation, où les parties matérielle et logicielle d'un système sont simulées conjointement pour valider leur fonctionnement et leurs interactions.

L'injection de fautes en simulation consiste à simuler le comportement du circuit en présence du modèle de faute choisi (inversion, collage, etc.). Il y a deux méthodes d'injection de fautes : soit on modifie la description du circuit pour émuler les fautes, soit on utilise les commandes du simulateur.

Lorsque l'on modifie la description du circuit, un ou plusieurs composants d'injection de fautes peuvent être ajoutés [JAR+94, FMR06] ou les changements peuvent être appliqués individuellement sur les composants [JAR+94, LH00, ZME03]. En pratique, le circuit est simulé de la même manière avec ou sans faute, à l'exception du fait qu'un signal permet d'activer les composants d'injection.

L'injection de fautes en utilisant les commandes du simulateur a l'avantage de ne pas modifier le circuit. Les outils d'injection en simulation utilisent un modèle du circuit (VHDL, Verilog, analogique ou spécifique à l'outil) et un modèle de faute. La simulation est lancée sur le modèle de circuit. Lorsqu'on atteint le moment d'injection, la (ou les) fautes sont injectées en modifiant la représentation de l'état interne du circuit (valeur des bascules, valeur des signaux, etc.). La simulation continue et l'évolution de l'état du circuit après l'injection de la faute et le résultat final sont analysés pour diagnostiquer le comportement du circuit.

FOCUS [CI92] a été développé en 1992 à l'université de Urbana-Champaign (Illinois). Il permet d'évaluer la sensibilité des fautes dans un circuit VLSI par simulation.

FOCUS injecte les fautes au niveau « device level » pendant l'exécution et reporte les conséquences au niveau porte (« gate level ») ou au-dessus. Pour cela, il utilise le simulateur SPLICE[2] « mixed-mode » analogique/numérique. Ce type d'analyse est possible grâce aux possibilités de simulation multiniveaux sur les circuits.

Les fautes sont injectées au niveau physique, en modifiant le voltage d'une interconnexion entre des éléments électriques et/ou logiques. Elles peuvent être simples ou multiples, temporaires ou permanentes. La forme d'onde de l'impulsion de courant appliquée à un nœud est programmable, ce qui permet de modéliser une augmentation de courant, une perturbation par particule alpha, un arc électrique et des fautes d'interconnexions.

Cette modélisation analogique est très proche du circuit mais elle nécessite beaucoup de temps. Une modélisation numérique est simulable plus rapidement.

MEFISTO (Multi-level Error/Fault Injection Simulation TOol) [JAR+94] injecte des fautes en utilisant des saboteurs (qui modifient la valeur et/ou le timing des signaux) et des mutants (altération ou remplacement de composant) pour injecter des fautes dans des descriptions hiérarchiques en VHDL.

Cet outil a été développé en 1994 conjointement au LAAS-CNRS à Toulouse et à l'université de Chalmers. MEFISTO permet d'injecter des fautes en modifiant le code VHDL et en utilisant les commandes internes du simulateur. C'est une des plus importantes contributions initiales au domaine puisqu'il a permis de simuler des fautes multi-niveaux sur une modélisation en VHDL.

L'utilisation des commandes du simulateur permet de simuler le composant pas à pas et de modifier les signaux et les variables pendant la simulation, pour un temps défini. Le simulateur utilisé n'est plus disponible (VHDL Optium de la société Viewlogic).

Le modèle de faute utilisé est complètement paramétrable. Il est appelé Abstract Fault Model (AFM). MEFISTO contient une base de données pré-définie d'AFM et il est possible de définir d'autres modèles en ajoutant des saboteurs et des mutants.

Pour le choix des cibles, 3 possibilités sont offertes : la sélection d'une cible, la sélection suivant une propriété ou la sélection au hasard. Ces critères peuvent être composés.

MEFISTO est outil d'injection de fautes puissant, mais il laisse le choix des fautes à l'utilisateur. De plus, sa dépendance par rapport à un simulateur l'a rendu inutilisable.

ASPHALT [YS96] a été développé à la Carnegie Mellon University en 1996 pour démontrer l'apport de l'utilisation d'un modèle de faute au niveau RTL (Register Transfer Level), qui est un compromis entre les fautes matérielles, couteuses en temps, et les fautes de plus haut niveau qui ne sont pas suffisamment représentatives des fautes matérielles actuelles.

[2] à ne pas confondre avec SPICE

ASPHALT utilise une description en RTL du circuit, sur lequel sont injectées toutes les fautes possibles définies dans le modèle (une par exécution). Grâce à une optimisation de la simulation, ASPHALT propose une accélération d'un facteur 512 par rapport à une description Verilog au niveau porte.

Seules les fautes intermittentes sont prises en compte puisque 80 à 90% des erreurs informatiques proviennent de fautes temporaires ou intermittentes. Cependant, cet argument n'est pas valide dans un contexte de sécurité, puisque l'attaquant choisit le type d'attaque dont il a besoin.

Le modèle de faute est très détaillé. Ses concepteurs le placent entre le niveau « micro-instruction » et le niveau « bit-flip ». Pour résumer, les fautes peuvent être sur les registres, les bus, les ALUs, le décodeur d'instruction ou les lignes de contrôle.

Les injections de fautes sont exhaustives et l'outil ne prend pas en compte la problématique de la sécurité.

DEPEND [GIY97] est un environnement de simulation fonctionnel basé sur des processus, développé en 1997 à l'University of Illinois at Urbana-Champaign. Il permet la modélisation des comportements des sytèmes complexes, ainsi que les dépendances entre composants.

L'approche est purement logicielle : le système est modélisé en utilisant les objets C++ de la bibliothèque DEPEND. Ces objets possèdent un comportement en faute intégré qui peut être changé. Le temps de simulation est réduit en utilisant la structure hiérarchique du circuit ainsi qu'en ignorant les fautes sans répercutions. Un des points intéressants est la réduction du nombre de fautes en se basant sur une analyse de la charge de travail.

Selon l'article, les principaux bénéfices sont l'utilisation d'un modèle comportemental et de scenarios de fautes réalistes. Cependant, le modèle de faute est défini dans la bibliothèque d'objets C++ et ne peut pas être affiné pour chaque instance de composant.

DEPEND simule les manifestations au niveau système des fautes au niveau porte. Il utilise des modèles de fautes fonctionnels car il se focalise sur le comportement des composants. Les fautes peuvent être temporaires ou non. Chaque composant est fourni avec un modèle de faute par défaut, qui peut être adapté par type de composant, mais pas par instance.

Le principal désavantage de DEPEND est sa modélisation en C++ qui n'est pas utilisée dans l'industrie du semiconducteur. Cependant, c'est l'un des rares outils à limiter le nombre de fautes en orientant l'utilisateur sur les fautes à effectuer.

VERIFY [STB97] (VHDL-based Evaluation of Reliability by Injecting Faults efficientlY) a été développé en 1997 à Erlangen en Allemagne pour évaluer la robustesse des systèmes numériques.

Le principe est d'étendre le langage VHDL décrivant les signaux et les composants pour prendre en compte l'injection de faute, ce qui nécessite un compilateur VHDL particulier. L'objectif est de permettre aux fondeurs de fournir des biblio-

thèques de conception contenant ces informations relatives aux fautes du composant.

Par rapport à MEFISTO, le modèle de faute est intégré au composant (tous les composants sont des mutants), ce qui permet d'injecter chaque faute sans devoir tout recompiler (ce qui est nécessaire avec les mutants de MEFISTO).

VERIFY utilise une méthode intéressante pour la simulation, qui consiste à ne simuler qu'entre l'injection de fautes et le moment où le système retrouve un fonctionnement normal (en ayant préalablement simulé un fonctionnement correct). C'est un outil qui permet de tester le taux de fonctionnement correct, puisqu'il compare la simulation du système sans faute avec le système avec faute.

Pour illustrer les possibilités d'injection de fautes, l'article montre comment on peut modéliser une porte NOT en ajoutant 2 signaux pour modéliser un stuck-at-0 et un stuck-at-1 de 5ns toutes les 20000 ou 30000 heures. La durée et le moment exact d'occurence de la faute sont déterminés par une méthode de Monte-Carlo, ce qui permet une prise en compte probabiliste du taux de défaillance pour la définition des fautes.

Le modèle de faute est limité au modèle pouvant être exprimé avec les signaux VHDL modifiés. Ce modèle de faute est basé sur les portes et l'outil ne permet pas les modèles plus généraux comme des injections multiples sur plusieurs portes au même moment.

SINJECT [ZME03] est outil d'injection de fautes développé à Téhéran en Iran en 2003 pour analyser la robustesse des systèmes numériques modélisés dans les langages de description de matériel usuels (VHDL, Verilog).

SINJECT injecte des fautes dans les descriptions mixtes VHDL et Verilog des circuits, même dans celles qui ne sont pas synthetisables. Cela implique d'utiliser un simulateur qui supporte les modèles mixtes. Pour chaque injection possible, il est nécessaire d'ajouter un signal supplémentaire aux portes correspondantes.

L'injection en Verilog peut être faite à plusieurs niveaux. Au niveau switch, la faute peut être un Transistor-Stuck-On, Transistor-Stuck-Off, Open fault et Short fault. Au niveau porte, on peut injecter un stuck-at-0, un stuck-at-1 ou un "gate remplacement" (par exemple, NAND to NOR). Au niveau RTL, les fautes possibles sont le bit-flip, le flip-to-1, le flip-to-0 et la micro-operation. Au niveau comportemental, le modèle est complet puisqu'il prend en compte les fautes de type Stuck-Then, Stuck-Else, Assignment Control, Dead Process, Dead Clause, Micro-Operation, Local Stuck-Data, Global Stuck-Data ainsi qu'une modification des compteurs de boucle. Au niveau structurel, l'injection de fautes se fait par remplacement d'un module par un autre.

L'injection en VHDL de faute se fait en utilisant les techniques de mutant et de saboteur identiques à MEFISTO.

Dans le cas des mutant, le circuit est modifié, ce qui l'éloigne du circuit final. De plus, le choix des fautes est laissé à l'utilisateur.

FITSEC [EMZ+03] est un outil d'injection de fautes développé à la Sharif Uni-

versity of Technology en 2003 dans le but d'accélérer les injections de fautes dans les circuits.

La simulation a le désavantage d'être relativement lente par rapport aux méthodes d'injection précédentes. De plus, certaines parties du circuit ne sont pas concernées par les injections de fautes.

FITSEC [EMZ+03] scinde le circuit entre une partie émulée sur FPGA, qui ne reçoit pas d'injection, et une partie simulée. Cela accélère les injections, puisque toute la partie émulée calcule beaucoup plus rapidement. Cette combinaison accélère les tests d'un facteur 100 par rapport à une simulation classique.

Cette approche est une solution à la lenteur de l'injection de faute en simulation lorsqu'une partie du circuit n'est pas concernée par les fautes.

L'injection en simulation se situe tôt dans le processus de conception. La modélisation du circuit est totalement flexible, ce qui assure une observabilité et une contrôlabilité totales, et permet une modélisation du système à plusieurs niveaux. De plus, cette technique a l'avantage d'être peu coûteuse puisqu'elle ne nécessite pas de matériel particulier.

Les outils d'injections en simulation fournissent une analyse très fine de la propagation d'une erreur dans le circuit, même s'ils dépendent de la précision des modèles fournis, parfois difficiles à obtenir (circuit, fautes, environnement opérationnel).

La complexité actuelle des circuits (nombre de portes, niveaux de métal, etc.) ne permet pas de simuler toutes les fautes possibles. Les circuits deviennent de plus en plus complexes, avec un nombre de portes logiques de plus en plus important, ce qui rend la simulation de toutes les fautes possibles très coûteuse en temps de simulation. Usuellement, un sous-ensemble des fautes possibles est choisi par l'utilisateur à partir de sa connaissance de la fonction du circuit (CPU, cryptographie, etc.).

L'injection en simulation est un domaine qui a été dynamisé par l'apparition de MEFISTO, injecteur de référence, en 1994. Le langage le plus représenté pour la modélisation du système est VHDL, probablement à cause de sa popularité dans l'industrie.

Nom	**Année**	**Modélisation**	**Niveau des fautes**
FOCUS	1992	VLSI	physique
MEFISTO	1994	VHDL	manipulation des signaux
ASPHALT	1996	RTL	niveau "assembleur"
DEPEND	1997	C++	comportement des composants
VERIFY	1997	VHDL	manipulation des signaux
SINJECT	2003	VHDL, Verilog	du niveau switch au niveau structurel
FITSEC	2003	VHDL, Verilog	du niveau switch au niveau structurel

1.3.4 Comparatif des méthodes d'injection de fautes

Pour tester le comportement réel d'un circuit soumis à des fautes, l'approche la plus immédiate est d'imiter l'environnement qui génère ces fautes. C'est le principe de l'injection physique qui reproduit les conditions physiques censées produire les fautes.

C'est la méthode la plus proche de la réalité, même si elle est seulement représentative des fautes de la méthode physique utilisée : les résultats d'une injection sur les interconnexions du circuit sont peu réutilisables pour savoir si le circuit est sensible au fautes par bombardement d'ions lourds.

Lors de l'analyse de l'effet des fautes sur le circuit, il est difficile d'obtenir les informations internes du circuit comme le lieu de création de la faute.

En plus de la construction du circuit, l'injection physique nécessite en général du matériel couteux pour injecter les fautes, ce qui en fait la méthode la plus couteuse en matériel.

L'injection logicielle répond à une partie de ces problèmes. Les fautes sont émulées en utilisant les mécanismes internes des circuits. Cela restreint les besoins matériels puisque tout est fourni par le circuit lui-même. Les fautes sont alors contrôlables et reproductibles, ce qui facilite l'analyse de l'effet des fautes.

Cependant, il est nécessaire de définir un modèle de faute à émuler représentatif des fautes contre lesquelles on souhaite se prémunir. Ce modèle est d'ailleurs limité par les capacités des mécanismes internes : on ne peut injecter des fautes qu'aux endroits accessibles en écriture à travers ces mécanismes et en particulier sur les objets manipulables par le logiciel.

L'injection en simulation va encore plus loin dans l'abstraction et utilise un modèle de circuit. Cette méthode est économique et il n'est pas nécessaire de construire physiquement le circuit. L'accessibilité et la controlabilité du circuit ne sont limitées que par les modèles utilisés.

En contrepartie, la pertinence des résultats est dépendante non-seulement du modèle de faute, mais aussi du modèle de circuit utilisé. Comme la simulation est beaucoup plus lente, il est possible d'utiliser une accélération matérielle en simulant une partie du circuit sur FPGA. Néanmoins, cette partie du circuit doit être soigneusement choisie, puisqu'elle sera émulée plus vite mais n'héritera pas des avantages liés à la simulation (controllabilité, accessibilité).

Pour conclure, l'injection de fautes physiques est utile dans le cas de circuits déjà réalisés physiquement et qui doivent être évalués dans un environnement physique connu, comme c'est le cas par exemple dans l'aéronautique.

L'injection en simulation est la situation la moins onéreuse, mais elle nécessite des modèles à la fois précis et complets. Cela en fait un bon candidat pour l'évaluation des circuits avant la construction de circuits, où seul un modèle est disponible.

L'injection logicielle est limitée à la fois par la portée des outils d'injection et par le modèle de faute choisi. Dans certains cas précis, notamment lorsque le modèle est similaire à celui utilisé en simulation, cette méthode permet d'obtenir des résultats comparables à ceux de la simulation, mais plus rapidement.

1.4 Synthèse

Alors que les contraintes industrielles et financières sont de plus en plus fortes, notamment en ce qui concerne le coût de production et le temps de mise sur le marché, la sécurité apparaît comme une contrainte supplémentaire à prendre en compte lors du processus de conception et de validation.

Cela est encore plus vrai dans le cas des cartes à puce, où c'est le circuit qui assure la sécurité d'un système plus vaste et plus complexe.

Une vue d'ensemble des algorithmes usuels de cryptographie a été présentée, avec la description en détail d'algorithmes symétriques et asymétriques.

Même s'ils sont réputés sûrs, leurs implémentations physiques peuvent être attaquées pour obtenir tout ou partie des informations sensibles qu'elles renferment. La synthèse des attaques classiques sur les circuits a permis de montrer les méthodes connues pour mettre en défaut leurs protections.

Enfin, une sélection des outils d'injection de fautes de référence a été fournie. Les trois méthodes principales d'injection sont l'injection physique, qui reproduit physiquement les fautes sur le circuit, l'injection logicielle, qui s'appuye sur les possibilités d'accès internes (comme le JTAG) pour reproduire l'effet de fautes et l'injection en simulation qui utilise une modélisation du circuit et des fautes.

La comparaison des avantages et des inconvénients des outils nous a permis de positionner notre approche par rapports aux outils existants. Notre but étant d'évaluer *a priori* la sécurité des circuits, seule la simulation est convenable pour notre analyse.

Chapitre 2

Méthodologie d'évaluation de la robustesse des circuits contre les attaques par faute

Compte tenu de la problématique de plus en plus importante de la sécurité des circuits ainsi que l'état de l'art actuel des méthodes d'injection de fautes, il est nécessaire de proposer une nouvelle manière d'appréhender la conception de circuits sécurisés.

Les contraintes de sécurité sont connues dès le début du flot de conception, ce qui permet de les prendre en compte dès l'étape de simulation fonctionnelle. Le recours à la simulation nécessite l'utilisation de modèles de faute qui soient réalistes. Ces modèles conditionnent la précision et la pertinence des résultats : un modèle trop précis augmentera le temps de simulation, alors qu'un modèle trop abstrait ne fournira pas toutes les informations nécessaires.

La méthodologie décrite dans ce chapitre doit être générique pour s'appliquer à un grand nombre de circuits de sécurité, tout en étant assez flexible pour réaliser des injections et des analyses spécifiques à un modèle de fautes et à un circuit particulier.

Dans ce chapitre, nous exposerons d'abord les objectifs de nos travaux par rapport à l'état de l'art actuel.

Cela nous conduira à définir plus précisément la problématique globale de l'évaluation des circuits par simulation de l'injection de fautes.

Ensuite, les phénomènes physiques correspondants aux modèles fautes possibles seront analysés pour extraire des modélisations logiques que nous utiliserons pour accélérer la simulation des fautes.

Compte-tenu de ces modèles de faute, les critères du choix des fautes à simuler seront présentés et donneront lieu à la notion de poids d'une faute, qui correspond à une métrique que l'on utilisera ensuite pour ordonner les injections de fautes.

Enfin, nous terminerons ce chapitre par les métriques utilisées pour évaluer la

sécurité des circuits en définissant les mesures utilisées.

2.1 Objectifs

Pour proposer une approche innovante, il est nécessaire de comprendre les approches existantes pour en améliorer certains aspects.

L'objectif de cette thèse est d'estimer la robustesse des circuits face aux attaques par faute en intégrant un outil d'aide à la conception dans le flot existant. Cela nécessite de proposer une nouvelle méthodologie de conception pour prendre en compte ces contraintes de sécurité.

2.1.1 Estimation de la robustesse face aux attaques par faute

Le contexte de ce travail est différent de ceux de outils présentés précédemment, puisque l'objectif n'est pas d'évaluer le sûreté de fonctionnement des circuits, mais leur sécurité. Contrairement au problème de la sûreté de fonctionnement où l'occurence de faute suit des lois physiques ou un processus accidentel dans le cas d'erreurs provoquées par du logiciel, l'injection de fautes est piloté par un attaquant qui cherche à obtenir des informations.

La sécurité des circuits impose de nouvelles contraintes, dans la mesure où le système est dans un environnement bien souvent entièrement contrôlé par l'attaquant et qu'il faut prendre en compte cela dès la conception du circuit.

Pour insister sur la notion de sécurité, notre cas d'étude est un circuit cryptographique d'AES. Plusieurs attaques par faute ont été publiées sur cet algorithme et notre démarche permettra de déterminer si ces attaques se retrouvent sur l'implémentation.

2.1.2 Choix d'un modèle de faute réaliste

Dans un souci de réalisme, notre approche consiste à utiliser une description du circuit non-modifiée sous la forme d'une interconnexion de composants de base (*netlist*). L'utilisation d'une netlist aplatie du circuit permet d'analyser et d'injecter des fautes de la manière la plus proche du comportement physique effectif, tout en restant dans la perspective d'une utilisation industrielle. Ce choix sera justifié plus en détails dans la section 3.2.1.

Par conséquent, le modèle de faute choisi pour valider l'approche doit aussi correspondre à des considérations physiques réalistes. La méthodologie sera extensible à d'autres modèles de faute, mais l'objectif principal est d'évaluer la sécurité du circuit face à des attaques concrètes, ce qui impose l'utilisation de modèles ayant une justification physique.

Cela permettra notamment de voir dans quelle mesure le modèle physique choisi correspond aux modèles utilisés dans les attaques par faute connues.

2.1.3 Méthodologie d'analyse et d'évaluation

Intégrée très tôt dans la conception du circuit, notre méthodologie utilisera l'injection de fautes en simulation. Cette approche peut être utilisée avant la réalisation physique du circuit, sans modifier sa description, tout en générant des fautes reproductibles, ce qui facilite l'analyse des résultats.

Comme la simulation nécessite beaucoup de temps, une analyse préalable du circuit réduira le nombre d'injection en se basant sur une analyse pertinente du circuit et du modèle de faute. Pour cela, l'outil fournira à l'utilisateur des informations sur les injections les plus importantes par exemple en prenant en compte la structure du circuit et son comportement en l'absence de fautes.

La campagne d'injection est l'étape la plus longue et la plus répétitive, il faudra donc l'automatiser entièrement pour fournir les résultats sans intervention de l'utilisateur, tout en veillant à recueillir les informations nécessaires à l'analyse souhaitée.

Finalement, la synthèse des résultats de simulation devra être adaptée aux critères de sécurité des circuits. En particulier, les métriques utilisées devront tenir compte du type de circuit et de ses protections.

2.2 Problématique globale

Lorsqu'un circuit est soumis à des attaques physiques, ces perturbations affectent son calcul, voire son résultat. Les fautes injectées peuvent modifier l'état interne du circuit en ayant un effet local ou global.

Le modèle de circuit utilisé dans notre travail est une interconnexion de bascules et de portes logiques. La première étape est de modéliser au niveau du circuit les effets de la faute, qui seront donc émulés au niveau logique. Il faut donc analyser les effets physiques produits pour modéliser les fautes au niveau logique.

L'émulation des fautes pourra aussi permettre de simuler des modèles plus complexes. Par exemple, on pourra émuler un modèle de faute multi-bits par plusieurs injections de faute mono-bit.

Idéalement, pour évaluer le robustesse d'un circuit face aux attaques par faute, il faudrait simuler l'effet de l'ensemble des fautes possibles sur le circuit.

Dans le cas de fautes qui ne modifient qu'un seul signal, le nombre de simulation est égal au nombre total de signaux multiplié par le nombre de cycle du circuit, ce qui est déjà irréalisable en pratique la plupart du temps. Il est alors inconcevable de faire l'analyse de toutes les fautes possibles, qui nécessiterait un nombre prohibitif de simulations.

Pour évaluer le circuit, il faut alors réduire significativement le nombre d'injections de faute. Plusieurs possibilités permettent de réduire le nombre de simulations nécessaires.

Tout d'abord, il faut ignorer les injections qui n'auront aucun impact sur le circuit. Certaines fautes ne durent pas assez longtemps pour être mémorisées dans l'état interne du circuit. La majorité des fautes injectées sur les nœuds du circuit n'ont pas réellement d'impact sur le circuit lorsqu'elles ne sont pas capturées par les éléments mémorisants.

Le circuit, même s'il a été physiquement perturbé, continue à fonctionner correctement. Il est donc possible d'éliminer une partie des fautes qui n'ont pas d'impact sur les bascules du circuit [GS95]. Par exemple, cela peut être le cas lorsque le circuit est soumis à des radiations d'intensité trop faibles ou lors de l'ajout d'un délai dont la durée n'est pas suffisante.

Dans le cas de fautes qui imposent une valeur à un signal (fautes de « collage »), le circuit ne sera pas impacté si la valeur imposée est celle que doit avoir le signal naturellement. Pour maximiser les chances de produire une erreur, il faut donc injecter des fautes qui dépendent de la valeur du signal, pour être sûr de changer sa valeur.

Toutes les parties d'un circuit ne sont parfois pas toutes actives au même moment. Dans le cas des fautes à portée locale, il convient d'écarter les fautes injectées dans une partie inactive puisqu'elle provoquera une erreur qui sera ensuite écrasée par des données valides [LH92].

De plus, il est possible de réduire le nombre de fautes injectées en prenant en compte un modèle de faute physique existant en pratique. Cela permet de se retreindre aux fautes correspondant à un certain type de perturbations physiques. Par exemple, certains types d'attaques physiques peuvent avoir un impact global sur plusieurs points du circuit : il n'est pas nécessaire de simuler les fautes très localisées.

Enfin, l'analyse du circuit doit permettre de déterminer la couverture du circuit face aux attaques par faute et fournir une quantification de la sécurité des circuits.

Cela nécessite de définir des métriques pour pouvoir estimer la sécurité d'un circuit, mais aussi pour comparer plusieurs circuits entre eux. Il sera alors possible d'évaluer l'apport de contremesures sur des circuits protégés. Ces critères de sécurité des circuits dépendent du type de circuit, ainsi que de ses propriétés de sécurité : protections, algorithme, attaques par faute connues.

2.3 Modèles de faute

Différents modèles de fautes peuvent être utilisés pour évaluer un circuit. Dans un premier temps, la terminologie autour des modèles de faute sera précisée. Après une présentation des fautes permanentes, nous expliciterons plus en détail les modèles de fautes transitoires, à travers les modèles que nous avons retenus pour notre étude.

2.3.1 Terminologie

Lors de l'utilisation d'un système, l'utilisateur cherche à exploiter le fonctionnement attendu du système. On parle de **sûreté de fonctionnement** d'un système quand celui-ci fournit un service dont les modalités s'intègrent dans le cadre convenu avec les systèmes (humains ou non) qui interagissent avec lui.

Ce cadre est formalisé dans la description d'une **spécification**. Un système dont le service dévie de sa spécification est dit **défaillant** : la confiance des utilisateurs dans ce système est remise en cause.

Un système défaillant est dans un état incorrect, appelé **erreur**. Le passage d'un état correct à un état erroné est dû à une **faute**. Une faute peut être interne au système : la conception originelle ne suit pas les spécifications ou le système s'est dégradé physiquement. Elle peut aussi être d'origine externe : l'effet de l'environnement sur le système peut avoir des conséquences sur le système.

Pour assurer le fonctionnement correct d'un système, il est donc nécessaire de se protéger des fautes, qui engendrent des erreurs, responsables de la défaillance du système.

La **tolérance aux fautes** permet au système de continuer à fournir un service conforme à la spécification malgré l'apparition de fautes. C'est la redondance de certaines fonctions qui prévient la transformation de la faute en erreur.

Lorsque des fautes apparaissent dans un système, on peut minimiser leur nombre et réduire leur impact : c'est l'**élimination de fautes**. Par exemple, on peut cloisonner les diverses fonctions d'un système pour éviter une trop grande propagation d'une erreur.

Lorsque l'on connaît le type, le nombre ou les conséquences des fautes, on tente de faire de la **prévision de fautes**, pour adapter les protections du système.

Des définitions plus détaillées de ces termes et notions, ainsi qu'une taxonomie complète pourront être trouvées dans [ALRL04].

2.3.2 Fautes permanentes

Les fautes permanentes sont la conséquence d'un problème de conception ou d'une dégradation du système.

Lors d'une faute de conception, le système est, par construction, légèrement différent de sa spécification. Ainsi, tout ou partie du système ne fournit pas le service attendu.

Une faute de conception peut être corrigée en modifiant ou remplaçant le système pour le rendre conforme à la spécification. Il est également possible de modifier la spécification pour prendre en compte le fonctionnement du système. Ce dernier cas est plus rare puisque cela implique de modifier les systèmes qui interagissent avec lui pour prendre en compte la modification de l'interface.

Pour se prémunir des fautes de conception, des méthodes de vérification formelle permettent de prouver un certain nombre de propriétés du circuit.

De plus, on utilise des données de tests pour valider le fonctionnement du système sur un sous-ensemble représentatif des données possibles. Par exemple, les circuits microélectroniques sont testés avec des données choisies pour vérifier que toutes les connexions internes ont bien été faites.

La dégradation d'un système au cours du temps survient lorsque qu'il y a modification des propriétés physiques des matériaux utilisés. Par exemple, le vieillissement thermique d'un semi-conducteur peut causer un court-circuit l'ouverture d'une de ses jonctions, voire des modifications des délais de propagation des signaux ou des niveaux de seuil.

Ces fautes surviennent après un certain temps de fonctionnement correct et sont irréversibles la plupart du temps.

Notre travail ne prends pas en compte les fautes permanentes puisque nous nous intéressons aux attaques différentielles par faute qui nécessitent de pouvoir obtenir également des résultats corrects du circuit, ainsi que des résultats perturbés par des fautes différentes.

2.3.3 Fautes transitoires d'inversion (bit-flip)

Les fautes transitoires sont plus difficiles à reproduire et donc à comprendre, puisqu'elles ne se produisent que dans certaines conditions. Bien souvent, lorsque l'effet des fautes transitoires apparaît, il est déjà trop tard pour essayer de retrouver la faute correspondante.

L'objectif d'utiliser un modèle de faute réaliste nécessite de considérer ces fautes au niveau physique pour pouvoir les modéliser correctement au niveau numérique.

Les deux modèles physiques considérés sont les attaques lasers, qui ont un effet local, et les impulsions sur le courant d'alimentation, dont l'effet est global.

2.3.3.1 Phénomène physique d'inversion

Le SEU (Single Event Upset) [Nor96] est un changement d'état créé par un effet d'ionisation dû au passage d'une particule chargée dans un système microélectronique. Ce phénomène est la cause du changement d'état d'un point mémoire en son inverse, c'est-à-dire un « bit-flip » (aussi appelé « upset »).

La particule produit sur son passage des paires électrons-trous qui produisent une impulsion de courant. Si son amplitude est suffisante et que l'effet dure assez longtemps pour que son effet soit mémorisé, alors la faute va changer l'état de la bascule.

Ce phénomène n'altère pas le circuit et est réversible, c'est-à-dire que la valeur modifiée pourra être corrigée lors de la prochaine écriture.

À l'origine, les SEUs ont été observés dans le domaine aérospatial [OBF+93]. Ils étaient provoqués par les radiations d'ions lourds provenant du soleil. Avec l'augmentation de la densité d'intégration des composants, les circuits sont plus sensibles aux SEUs qui peuvent être provoqués par des protons et des neutrons atmosphériques.

Un phénomène d'inversion statique d'une bascule, c'est-à-dire indépendant de l'horloge, peut aussi être provoqué par une impulsion laser. Les apports d'énergie que peut produire un laser imposent une valeur à certains signaux, ce qui a le même effet qu'une inversion [BECN+04].

2.3.3.2 Modélisation logique

La modélisation correspondante est une inversion logique d'une ou plusieurs bascules du circuit. Ce modèle de faute, appelé « bit-flip », est utilisé dans un grand nombre d'attaques par faute [Gir04, CT05].

Comme l'effet est local, la multiplicité des fautes est faible, même si elle ne peut être déterminée précisément que par des expérimentations physiques puisque beaucoup de paramètres rentrent en compte, notamment la technologie utilisée par le circuit et la précision du faisceau laser.

Il est important de préciser que ce modèle ne représente que les perturbations qui durent moins d'un cycle, puisqu'au niveau physique, un SEU va générer la modification d'une tension dans un seul sens. Cela signifie qu'une perturbation physique qui touche deux cycles et qui fait passer un signal de 1 à 0 dans le premier cycle, continuera à forcer cette valeur à 0 dans le deuxième cycle.
Un bit-flip produit par une impulsion laser ne perturbera la valeur de la bascule que pendant un cycle d'horloge.

Le modèle du bit-flip désigne donc un modèle de faute transitoire d'une durée maximum d'un cycle et qui modifie de façon certaine la valeur d'un bit du circuit.

2.3.4 Fautes transitoires de délai

Dans les circuits synchrones, l'état du circuit pendant le calcul est mémorisé dans l'ensemble de ses bascules. Chaque bascule n'est pas mise à jour instantanément lors du front d'horloge : un signal a besoin de temps pour se propager à travers les portes logiques vers les bascules. Ainsi, les parties combinatoires prennent du temps pour effectuer leurs calculs. Le cône logique d'une bascule est l'ensemble des portes logiques qui définissent sa valeur.
Entre le début du cycle d'horloge et la fin du délai de traversée du cône logique, le résultat du cône logique fluctue (figure 2.1).

C'est pour cette raison que la période d'horloge doit être supérieure au temps de traversée de tous les cônes logiques. C'est le chemin critique (celui qui a le temps de traversée le plus long) qui définit la fréquence maximum de fonctionnement d'un circuit. La différence entre le cycle de l'horloge et le délai d'un cône logique (temps maximum de traversée) constitue une marge de délai.

Le temps de propagation d'une porte dépend de plusieurs paramètres parmi lesquels la température et le niveau d'alimentation. Une valeur inattendue d'un de ces paramètres peut induire un délai qui rendra le temps de propagation total plus long qu'un cycle d'horloge. Après un front d'horloge, la valeur de sortie de la partie logique peut varier mais doit être stable quand le prochain front d'horloge a lieu. Si un délai est ajouté, la sortie peut être dans un état instable incorrect au moment de la mémorisation de certains cônes.

Les fautes de délai transitoires correspondent à des perturbations temporaires (fluctuation de l'alimentation, interférence électromagnétique, radiations) qui produisent des mémorisations de valeurs erronées dans certaines bascules. Les circuits actuels sont plus sensibles à cause de la miniaturisation et des plus faibles charges nécessaires pour représenter un état logique haut (HIGH).

L'un des objectifs de cette thèse est d'utiliser un modèle de faute ayant une justification physique (comme c'est le cas dans le domaine de la sûreté de fonctionnement), plutôt qu'un modèle adapté aux attaques par faute (comme c'est le cas en cryptographie).

Le modèle de faute de délais regroupe plusieurs possibilités de fautes physiques. En effet, un délai peut être ajouté à un circuit lors du passage d'une particule, mais aussi lorsque l'alimentation du circuit est perturbée pendant un court laps de temps (*power glitch*), par exemple.

2.3.4.1 Phénomène physique de délai

Les SETs (Single Event Transient) sont des impulsions résultant d'un passage d'une particule chargée dans la logique combinatoire d'un circuit.

Le phénomène physique est similaire aux SEUs, mais il se manifeste par une impulsion de courant dans un nœud de la glue logique, et pas forcément par une inversion. Cette impulsion perturbe la valeur d'un signal et peut ajouter un délai de propagation supplémentaire. Si ce délai est suffisant, la mémorisation suivante va capturer une valeur incorrecte (figure 2.2), ce qui correspond à un effet similaire à celui induit par un SEU. Le noeud perturbé peut être partagé par plusieurs cônes logiques, ce qui se traduira par la modification de plusieurs bits au niveau logique.

Comme l'impulsion transitoire doit être propagée jusqu'aux bascules, il est possible qu'elle soit atténuée, voire éliminée avant d'arriver à la fin du chemin de pro-

pagation. Cependant, les portes logiques deviennent très rapides et les temps de propagation très courts, ce qui tend à réduire cette atténuation [Nic99].

Dans le cas d'une perturbation de tension sur l'alimentation électrique, un délai temporaire de propagation des signaux est ajouté au circuit entier.

Physiquement, une baisse de la tension d'alimentation diminue les tensions internes du circuit ce qui augmente les temps de transitions des composants. Le seuil de commutation des portes est atteint plus tard et le traitement nécessite un temps additionnel.

Le délai du chemin critique peut alors être dépassé ce qui provoque des erreurs. Le circuit calcule des résultats faux (Cf. figure 2.3) qui peuvent être exploités par un attaquant.

Un circuit statique (dont les signaux ne changent pas) n'est pas vulnérable, puisqu'il n'y a pas de transitions à ralentir.

L'effet est global puisque toute la logique du circuit est impactée par la baisse de tension d'alimentation. Une quantité de délai est ajoutée à chacun des chemins de propagation des signaux.

Dans les deux cas, la valeur du délai ajoutée est importante puisque c'est elle qui détermine les bascules qui vont être affectées. Suivant le délai, toutes ne seront peut être pas touchées, et certaines pourront même capturer une valeur correcte bien que le calcul ne soit pas encore terminé.

2.3.4.2 Modélisation logique

Une faute de délai peut apparaître dans une bascule lorsque le délai ajouté par la faute dépasse la marge de délai. Cela revient à définir un seuil de délai des cônes logiques à partir duquel les bascules correspondantes mémorisent un résultat intermédiaire.

Cependant, ce résultat n'est pas nécessairement erroné puisqu'il est possible que la valeur instable du signal de sortie du cône logique soit celle qui doit être mémorisé finalement.

Lorsqu'un délai est dépassé et qu'une valeur erronée est mémorisée sur un bit, son effet est équivalent à un bit-flip à l'entrée de la mémorisation, au moment du front d'horloge. En effet, nous considérons les fautes transitoires qui affectent le comportement du circuit, c'est-à-dire celles qui se manifestent en erreur sur l'état du circuit.

Pour modifier l'état d'un circuit, ces fautes doivent être capturées par l'état du circuit, c'est-à-dire par ses bascules. En considérant une faute transitoire sur un seul bit, la faute peut être un collage-à-0, un collage-à-1 ou une inversion. Un collage-à-X ne modifie pas l'état d'une bascule qui est déjà à l'état X.

C'est pour cela que nous avons choisi le modèle de l'inversion. On peut alors émuler ces fautes en forçant la valeur du signal d'entrée d'une bascule à sa valeur

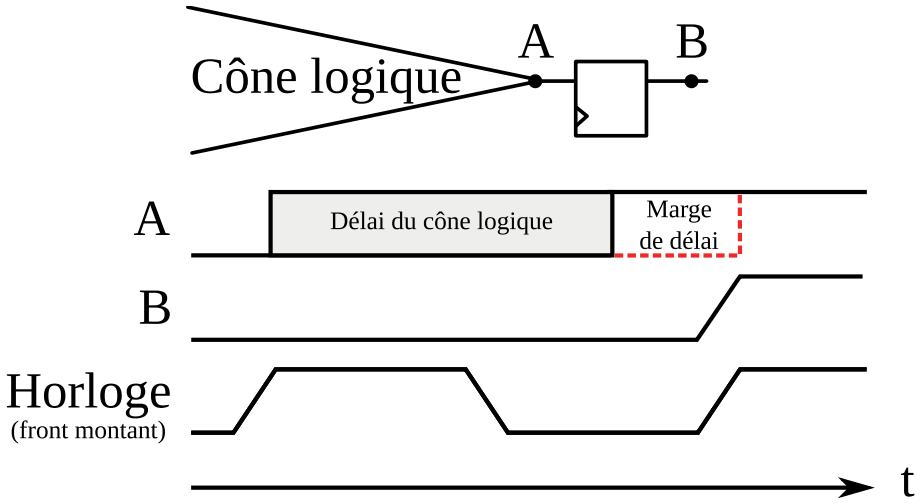

FIG. 2.1 – Délai d'un cône logique et marge de délai

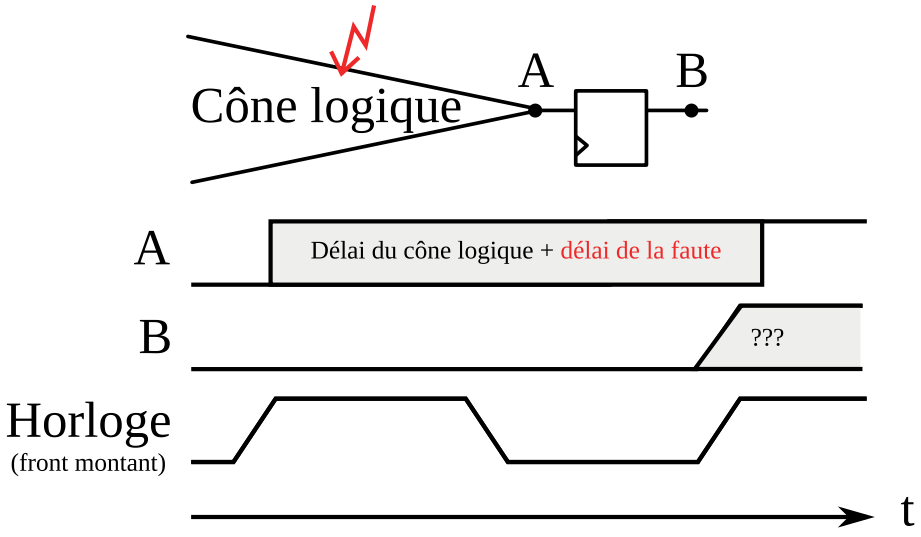

FIG. 2.2 – Délai d'un cône logique et faute de délai

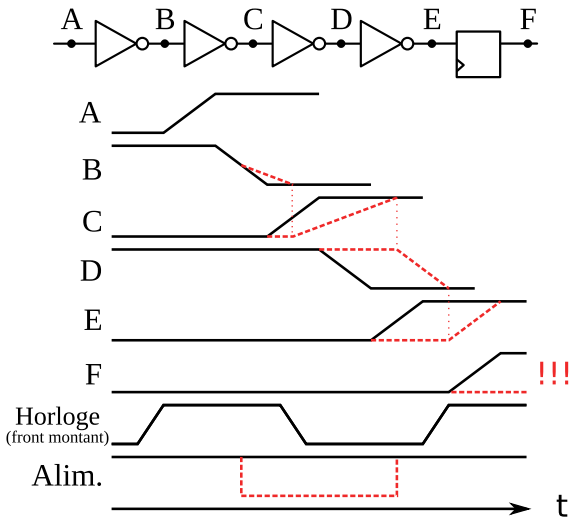

FIG. 2.3 – Exemple d'un circuit et d'un ajout de délai qui provoque une erreur

inverse pendant le front d'horloge.

Dans le contexte de notre étude, l'élimination des fautes ne produisant à coup sûr aucune erreur permet de concentrer l'analyse sur celles qui ont un meilleur potentiel pour la réalisation d'attaques par faute.

2.4 Choix des fautes à simuler

Même en limitant nos injections aux fautes des modèles numériques, leur nombre est encore trop élevé pour en simuler l'intégralité. Pour réduire encore le nombre d'injections, il est nécessaire de s'intéresser à d'autres paramètres.

En s'appuyant sur les modèles de faute, notamment la localisation, le moment et la multiplicité des fautes, il est possible de sélectionner les fautes les plus probables.

2.4.1 Modèle de faute d'inversion

La localisation de ce type de faute est très ciblée : l'injection ne concerne que les bascules du circuit. Cependant, elles sont toutes vulnérables, ce qui représente une grande quantité d'injection sur les circuits ayant des bancs de registres de grande capacité.

Pour de petits circuits, l'injection exhaustive de fautes simples est encore faisable en temps raisonnable, puisqu'elle nécessite un nombre de simulations linéaire en nombre de top d'horloge comme en nombre de bascules.

Pour pouvoir considérer des injections multiples, il faut restreindre l'analyse par d'autres critères de choix, puisque le nombre d'injection augmente exponentiellement par rapport à la multiplicité des fautes.

Même si les critères issus du modèle de faute numérique ne sont pas suffisants, d'autres pistes sont possibles, notamment en exploitant la fonction du circuit comme nous le verrons dans le chapitre suivant, consacré à l'implémentation de l'outil et des méthodes de pondération.

2.4.2 Modèles de faute de délai

Avec une faute de délai, les seuls cônes qui peuvent être touchés seront ceux dont les marges de délai sont plus petits que le délai de la faute.

Le nombre de bascules perturbées dépend de la précision avec laquelle le délai est ajouté. Si un attaquant peut injecter de très petites quantité de délai, il va plus facilement toucher un nombre restreint de bascules c'est-à-dire celles qui ont les délais les plus longs et qui passent par un état instable peu avant la mémorisation. Cela peut être les chemins critiques mais aussi des chemins un peu moins longs qui sont dans un état instable au moment de la mémorisation. Si au contraire il injecte un délai élevé, il va perturber tout le circuit dans la plupart des cas.

Le modèle de délai dynamique que nous proposons permet d'éliminer les parties inactives du circuit et de déterminer avec plus de précision le délai d'une bascule lorsque peu d'entrées du cône logique correspondant changent de valeur.

Fautes simples de délai Pour que l'ajout d'un délai ait une conséquence sur l'état du circuit, il faut que la valeur de l'entrée de la bascule fluctue et qu'elle soit capturée à un moment où sa valeur est différente de sa future valeur stable. Pour cela, plusieurs conditions doivent être réunies : les valeurs des entrées du cône ont changées (ce qui crée la fluctuation) et le maximum des délais de propagation de ces entrées jusqu'à la sortie ajouté au délai induit est supérieur à la période du cycle d'horloge (la marge de délai est dépassée). La probabilité qu'une faute de délai se manifeste en une erreur est dépendante du délai du chemin de propagation à travers la logique combinatoire et de la taille du délai ajouté.

Il y a deux possibilités pour que l'ajout d'un délai ne perturbe qu'un seul bit de l'état du circuit avec certitude. Soit le délai n'a été ajouté que sur des portes logiques spécifiques à une seule bascule : c'est un cas possible avec un SET. Soit le délai ajouté n'a entraîné un dépassement du temps critique seulement pour un seul bit d'état : cela est possible avec une perturbation de la tension d'alimentation.

Dans le premier cas, les bascules vulnérables sont celles dont la valeur fluctue. Si la précision de l'ajout de délai est connue, il est possible de ne considérer que les bascules telles que le délai ajouté dépasse la marge de délai.

Dans le second cas, ce sont les bascules contenant le résultat des cônes dont les temps de traversée sont les plus longs qui sont le plus susceptibles d'être perturbées. Suivant la précision du délai ajouté, le nombre de bits touchés peut être réduit. Si un seul bit est touché alors que le délai a été ajouté sur chacune des bascules, cela signifie que ce bit avait le cône logique le plus long du circuit, ou qu'un certain

nombre de bascules ont été touchées, mais qu'une seule a mémorisé un résultat faux.

Fautes multiples de délai Une modification de plusieurs bits d'état apparaît lorsqu'une faute a lieu à un endroit de la logique combinatoire partagée par plusieurs bascules, ou lorsque l'ajout de délai touche plusieurs cônes logiques parce que leur délai est assez long.

Dans le premier cas, plusieurs bascules peuvent être touchées à la même injection seulement si elles partagent une partie de leur cône logique. La multiplicité des fautes peut alors être bornée par le nombre de bascules qui utilisent les mêmes portes. De plus, la probabilité que plusieurs bits donnés soient affectés par une faute est proportionnel au nombre de portes communes à leurs cônes respectifs.

Dans le cas d'un délai ajouté à tous les cônes du circuit, les cônes avec les délais les plus longs seront touchés en priorité. Les cônes qui vont potentiellement mémoriser une valeur fausse sont ceux dont le délai induit ajouté au délai dynamique dépasse le cycle de l'horloge. La multiplicité des fautes augmentera avec la durée du délai ajouté. Si les différences entre les délais des cônes logiques sont réduites et que la quantité de délai ajouté n'est pas très précise, alors les fautes seront immédiatement de grande multiplicité.

Cependant, dans les 2 cas, il n'y a aucune garantie que les bascules mémorisent effectivement des données erronées. Même si les sorties des cônes logiques sont instables, il est possible que certaines d'entre elles mémorisent des valeurs correctes. Pour tester tous les cas de figures lorsque n bascules peuvent être touchées, il faut considérer $2^n - 1$ modifications de l'état du circuit, puisque chacune des bascules peut être touchée ou non. Toutes ces possibilités de faute ne sont pas nécessairement réalistes, mais distinguer celles qui le sont nécessiterait une analyse encore plus fine et complexe du circuit au niveau analogique pour chacun des cônes logiques.

L'émulation des fautes multiples de délais au niveau numérique peut être effectuée en utilisant plusieurs bit-flips sur un ensemble de bascules. De la même manière qu'en utilisant les fautes simples, on force la valeur des signaux concernés pendant le front d'horloge pour émuler la mémorisation d'un état incorrect.

Les informations qui permettent de savoir où peuvent survenir les fautes multiples peuvent être obtenues par analyse statique de la description du circuit en utilisant les informations d'interconnexion et de temps de traversée générées lors de l'élaboration de la *netlist*.

Le temps de traversée des cônes logiques est donc le maximum des temps de propagation de chacune des entrées des cônes logiques. C'est une estimation qui est calculée dans le pire cas, c'est-à-dire qu'on considère que le temps de mise à jour de la valeur de sortie d'un cône logique nécessite, à chaque fois, le temps de propagation de l'entrée la plus éloignée.

2.4.3 Modèle de faute délai dynamique

Pour affiner encore le modèle de fautes de délai, il est possible de distinguer les cônes logiques dont les entrées ont changé. En effet, si les entrées du cône logique ne changent pas, l'ajout d'un délai ne va pas causer de fautes puisqu'aucune fluctuation de la valeur de sortie du cône logique n'aura lieu. Dans ce cas, le délai d'un cône logique est nul, puisque la valeur à calculer est déjà disponible.

Dans le même ordre d'idée, on peut réduire le délai d'un cône logique au maximum des temps de propagation des entrées qui commutent. En effet, si seule une entrée très proche de la sortie du cône logique est modifiée, le temps pendant lequel la sortie va être instable (et peut être erronée) sera le temps de propagation de cette entrée, et non le maximum du temps de propagation de toutes les entrées, comme le montre la figure 2.4.

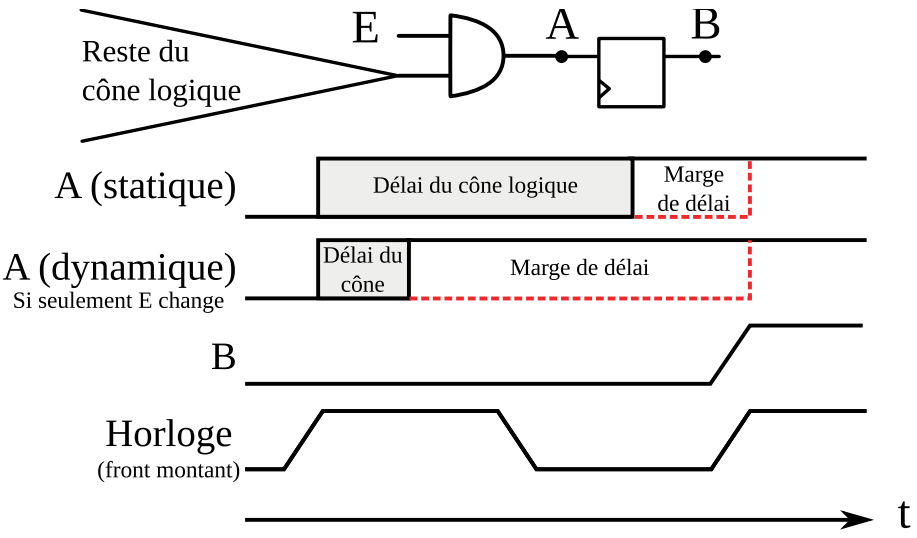

FIG. 2.4 – Délai dynamique

Cependant, l'analyse nécessaire à la prise en compte de cette amélioration devient dépendante des données, puisqu'il faut au préalable utiliser les données de l'exécution sans fautes pour savoir quelles sont les entrées qui commutent.

Par contre, la faute considérée ne doit toucher qu'un seul cycle du circuit. Dans le cas contraire, la modification produite au premier cycle risque d'annuler les commutations observées lors du deuxième cycle de l'exécution sans fautes, voire de créer de nouvelles commutations.

2.4.4 Notion de poids

Pour matérialiser le choix des fautes, nous avons choisi d'associer à chaque faute un poids, représentatif de l'intérêt de simuler cette faute pour effectuer l'analyse de

sécurité du circuit. C'est une manière d'ordonner les fautes par rapport au modèle choisi pour pouvoir sélectionner les injections.

Un poids sera en général associé à un endroit et un instant du circuit. Concrètement, cela signifie qu'un poids sera affectée à une bascule à un moment donné de l'exécution du circuit, indiquant l'intérêt porté à une faute sur cette bascule à cet instant. Si la dimension temporelle n'est pas prise en compte, le poids ne sera associé qu'à un endroit du circuit.

La formule de calcul du poids dépends naturellement du modèle de faute ainsi que de celui du circuit, mais aussi de la manière dont on veut sélectionner les fautes. Par exemple, on pourra affecter un poids nul à des fautes dont on sait qu'elles n'auront aucun impact sur le circuit et un poids élevé à des fautes que l'on veut tester à tout prix.

Dans le chapitre suivant (section 3.2.2), nous proposerons des pondérations adaptées à notre implémentation et aux modèles de fautes considérés.

2.5 Évaluation de la sécurité des circuits

Les métriques de sécurité des circuits doivent être définies pour prendre en compte les attaques par faute. Nous distinguons deux cas : le cas où le circuit n'est pas protégé et lorsque des protections, qui peuvent être des contremesures, ont été ajoutées au circuit. Cela permettra en outre d'estimer l'efficacité relative de plusieurs protections.

2.5.1 Circuit sans protection

La première information d'un circuit est d'abord son taux de résultats corrects. C'est le taux de résultat mathématiquement juste en sortie du circuit. Dans un contexte de sécurité, cela signifie de fournir le résultat attendu aux interfaces des autres parties du système.

Une définition plus souple est de considérer correct un résultat mathématiquement correct sans contrainte de temps. Ainsi, cela permet d'exécuter à nouveau un calcul que l'on soupçonne faux. Cependant, cela ouvre la porte à des attaques qui combinent les fautes et des attaques en temps (Cf. section 1.2.4.1).

Dans ce cas, même si les résultats fournis sont corrects, le comportement lors du calcul peut changer. En effet, faire un calcul exact, mais en perturbant le circuit, par exemple en changeant la consommation de courant ou le temps d'exécution, n'est pas acceptable puisqu'il devient vulnérable aux attaques décrites à la section 1.2.4. Il est donc possible de définir un taux de résultats corrects modifiant le comportement. C'est par exemple le cas d'une attaque en *safe error* sur un circuit qui corrige son résultat en recalculant : le circuit finira par fournir une valeur correcte, mais le calcul aura été deux fois plus long.

Après une perturbation, il est possible que l'état du circuit, devenu incohérent, ne lui permette pas de terminer son calcul. C'est par exemple le cas lorsque le circuit croit avoir déjà terminé son calcul ou s'il rentre dans un cycle de fonctionnement sans fin. Le taux de blocage d'un circuit est acceptable en terme de sécurité dans la mesure où aucune information n'est fournie à l'attaquant, à part le fait que le circuit a été perturbé.

L'hypothèse la plus probable lors de la perturbation d'un circuit est le calcul d'un résultat faux. L'état du circuit, une fois modifié, cause une erreur qui sera propagée jusqu'aux interfaces du circuit et exposée à l'attaquant.

Dans un contexte de sécurité, on cherche à évaluer la robustesse face aux attaques par faute. Ainsi, une sortie aléatoire est acceptable parce qu'elle ne fournit pas d'information à l'attaquant.

À l'inverse, lorsque l'algorithme du circuit est connu, certains résultats erronés peuvent être exploités en utilisant des attaques par faute connues, notamment pour les circuits de cryptographie. Le taux de résultats faux exploitables est un sous-ensemble du taux de résultats faux. Néanmoins, ce taux doit être considéré avec prudence dans la mesure où il ne représente que les attaques connues.

Les proportions de ces différents cas de figure permettront d'évaluer la sécurité intrinsèque des circuits. L'autre apport sera de déterminer le seuil de multiplicité des fautes à partir duquel le circuit se comportera de la même manière. En effet, la plupart des circuits réutilisent leur état interne au fur et à mesure de leur calcul : à partir d'un certain nombre de bits erronés, la propagation dans le circuit est comparable.

2.5.2 Circuit avec mécanisme de détection

Lorsque le circuit comporte des mécanismes de détection, d'autres métriques sont à prendre en considération pour caractériser leurs apports en terme de sécurité. Pour chacun des cas du circuit sans protection, on obtient une proportion d'exécution où le mécanisme de détection a été déclenché.

Tout d'abord, une faute qui n'a aucun effet ni sur le comportement ni sur le résultat du circuit ne devrait pas déclencher le mécanisme de détection. Sinon, cette détection sera considérée comme un faux-positif, ce qui peut être acceptable dans un cadre de sécurité. Ce comportement peut avoir lieu typiquement lorsque la faute touche le système de détection lui-même. Il faut cependant noter que si l'information de détection est transmise à l'extérieur du circuit, elle peut être considérée comme un changement de comportement et permettre l'élaboration d'attaques par faute, notamment avec le principe du *safe error*.

Les blocages peuvent être considérés comme des résultats corrects ou non, suivant

l'application visée.

En effet, le blocage d'un circuit peut être une protection efficace pour éviter de fournir un résultat faux : la simulation d'un circuit bloqué sera décompté comme s'il avait fournit un résultat correct.

À l'inverse, on peut vouloir aussi détecter les blocages, comme toutes les injections de fautes qui changent le comportement du circuit, même si elles ne produisent pas d'erreur.

L'efficacité de détection pourra être évaluée grossièrement en calculant le rapport entre les résultats faux détectés et le total des résultats faux. Dans le cadre des attaques par faute, il est important de mesurer le taux de résultats faux qui ne sont pas détectés.

En effet, ils peuvent alors être utilisés pour cryptanalyser le circuit, surtout si ces résultats faux sont exploitables avec des attaques connues. Un niveau supplémentaire peut être envisagé pour dénombrer quelles sont les injections ayant produit une erreur et qui sont exploitables avec les attaques par faute actuelles.

La classification des réactions d'un circuit face à des fautes est schématisé par la figure 2.5.

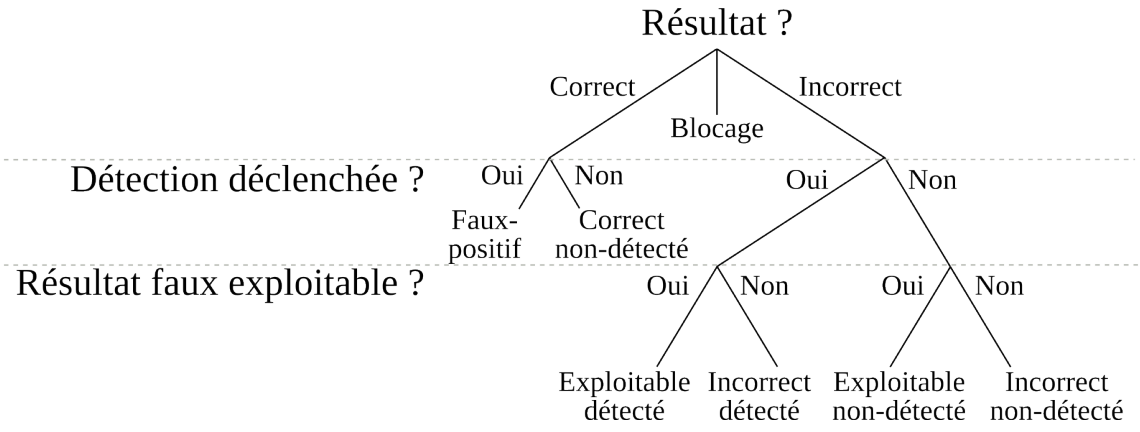

FIG. 2.5 – Analyse de sécurité d'un circuit avec mécanisme de détection

2.6 Synthèse

Dans ce chapitre, nous avons décrit la méthodologie d'évaluation de la sécurité des circuits face aux fautes que nous avons retenue pour nos travaux.

Les objectifs et la problématique globale ont d'abord été expliqués pour justifier le besoin d'un modèle de faute réaliste pour l'estimation de la robustesse des circuits face aux attaques par faute. Le nombre de fautes possibles étant trop élevé pour une

simulation exhaustive, il faut en sélectionner une partie représentative en prenant en compte le modèle de faute.

Nous avons choisi de nous intéresser aux perturbations physiques d'inversion et d'ajout de délai, que nous avons modélisées par des mémorisations erronées à l'entrée des bascules, pour pouvoir les simuler numériquement. Ces modèles correspondent à plusieurs perturbations physiques, notamment les SEU ou les perturbations de la tension d'alimentation, ce qui rejoint notre préoccupation d'utiliser des modèles de fautes proche de fautes physiques.

Les critères de choix des fautes se manifestent à travers un poids qui permet de classer les injections de faute.

Enfin, pour évaluer la sécurité des circuits, des métriques de sécurité ont été définies en distinguant le cas des circuits protégés et sans mécanisme de protection. En effet, une faute peut être ignorée par le circuit, le bloquer ou conduire à un résultat incorrect. Dans ce dernier cas, il est intéressant d'analyser quelles sont les fautes qui peuvent être exploitées pour contourner les protections du circuit, et si elles sont détectées par le système de détection, quand celui-ci est disponible.

Ces critères serviront à caractériser le niveau de sécurité des circuits et pourront être utilisés pour comparer la sécurité de plusieurs implémentations du même algorithme.

Cette méthodologie, résolument orientée vers le rapprochement des problématiques de sécurité et de conception des circuits, a été implémentée dans PAFI (*Prototype of Another Fault Injector*), un outil d'injection de fautes combinant l'analyse du circuit, l'automatisation de la campagne d'injections de faute et l'analyse des résultats, que nous présentons dans la section suivante.

Chapitre 3

PAFI : outil d'injection de fautes et d'analyse de la sécurité des circuits

Nous avons défini une méthodologie qui effectue une analyse préalable d'une modélisation du circuit en vue de fournir une évaluation de la sécurité des circuits. Cette méthodologie a été implémentée dans un outil appelé PAFI (*Prototype of Another Fault Injector*).

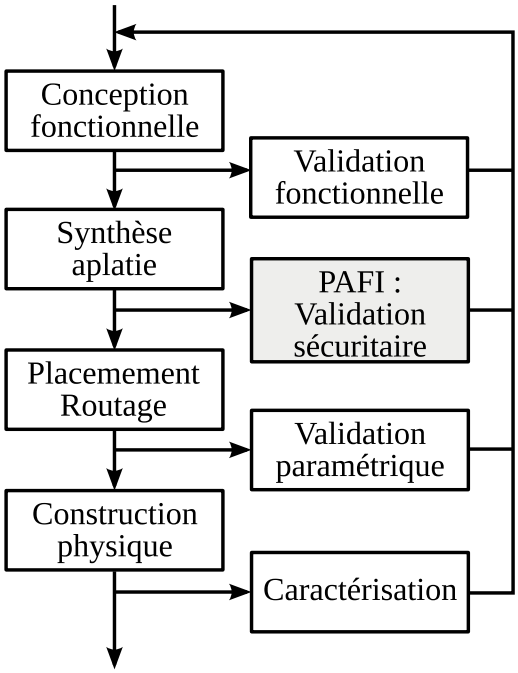

FIG. 3.1 – Intégration de PAFI dans le flot de conception

PAFI [FTF+07] s'intègre dans le flot de conception (figure 3.1) au niveau de

la validation sécuritaire, c'est-à-dire après la validation fonctionnelle mais avant la validation paramétrique, conformément au modèle de circuit utilisé.

Cela permet notamment de garantir le comportement et les fonctionnalités du système avant de prendre en compte les caractéristiques de sécurité. En effet, la garantie de respecter le cahier des charges fonctionnel prime sur les contraintes de sécurité.

À l'inverse, notre analyse de la sécurité se situe en amont du placement-routage. Ce choix a été fait pour ne pas pénaliser le temps d'analyse et de simulation, mais cette analyse aurait pu être faite après placement-routage, pour obtenir des résultats encore plus pertinents, au prix d'un temps de simulation important.

Comme défini dans notre méthodologie, PAFI utilise une modélisation logique sans hierarchie du circuit pour prendre en compte les détails précis sur le circuits (portes, délais).

Comme le nombre de fautes à simuler est grand, il est nécessaire de sélectionner les fautes les plus pertinentes pour l'analyse souhaitée.

L'aspect novateur de ce travail est de définir les critères de sélection des injections de fautes au regard d'un modèle de faute. Notre approche est d'aider l'utilisateur à sélectionner les fautes les plus significatives pour réduire le temps d'injection, ainsi que d'automatiser les campagnes de simulation.

L'objectif est de guider, *a priori*, la campagne d'injection vers des fautes qui ont une grande probabilité d'avoir un impact sur le comportement du circuit.

3.1 Flot de conception général

PAFI est un outil d'analyse de la fiabilité et de la sécurité des circuits par injection de fautes en simulation. Pour injecter les fautes, il utilise les commandes du simulateur et donc ne modifie pas la description du circuit. Son architecture générale est représentée sur la figure 3.2.

Comme PAFI est le résultat d'un travail de recherche, il a été conçu pour être modulaire. En effet, comme il est difficile de prédire en début de thèse le résultat final et que le sujet peut être réorienté, il est prudent de prévoir de changer certaines parties de l'outil.

De plus, cette séparation en plusieurs modules rentre dans le cadre des bonnes pratiques du développement logiciel : chaque module peut être testé, remplacé et optimisé indépendamment, la réutilisation est facilitée.

Par contre, cela implique de définir correctement le format des fichiers d'échange entre les modules. Si un nouveau modèle de faute nécessite un nouveau type d'injection de fautes, le format des fichiers de sortie doit être adapté en conséquence et cela à un impact sur le module d'injection. Pour cela, nous avons utilisé des formats XML : ils sont extensibles et peuvent être complétés au fur et à mesure. Ils ont

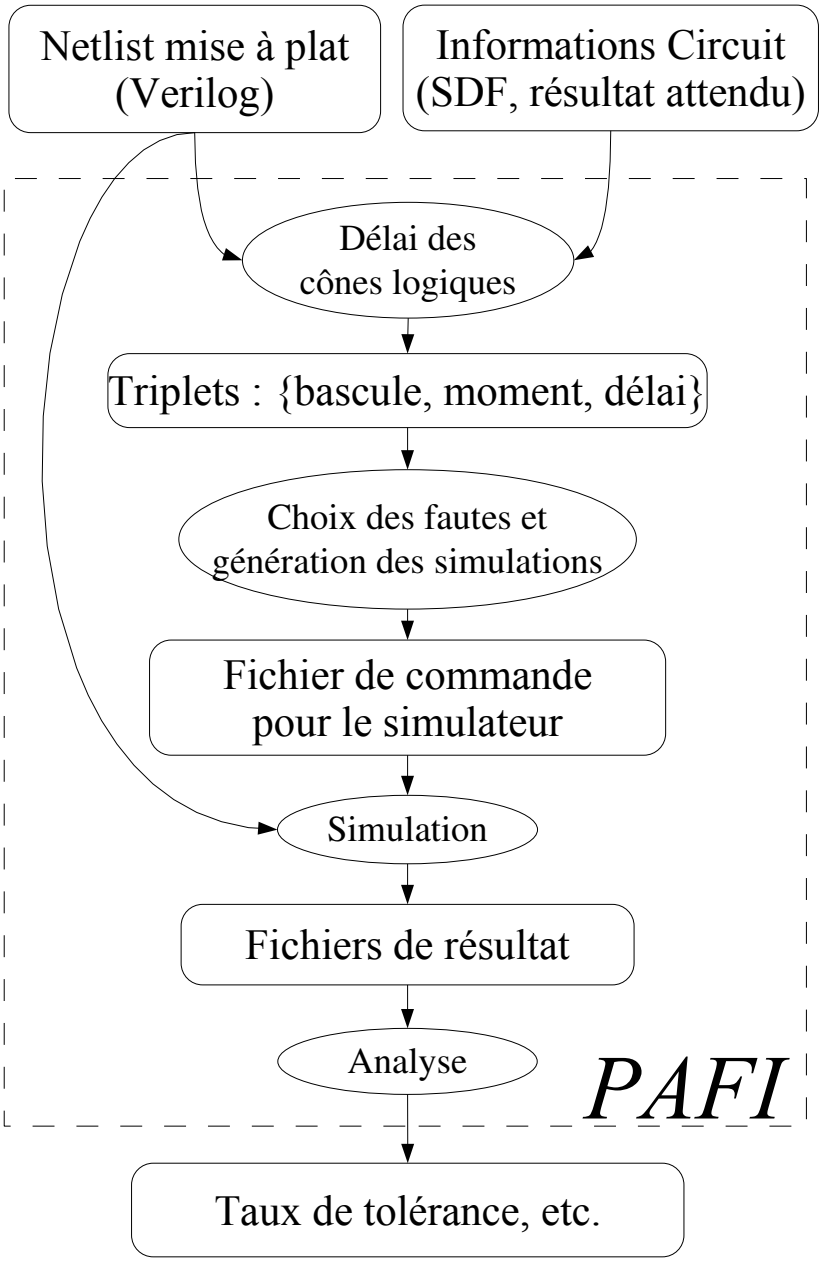
Netlist mise à plat
(Verilog)
Informations Circuit
(SDF, résultat attendu)
Délai des
cônes logiques
Triplets : {bascule, moment, délai}
Choix des fautes et
génération des simulations
Fichier de commande
pour le simulateur
Simulation
Fichiers de résultat
Analyse
PAFI
Taux de tolérance, etc.

FIG. 3.2 – PAFI

l'avantage d'être lisibles par l'utilisateur (format texte) et facilement analysables par la plupart des langages modernes.

Chaque étape est effectuée par un programme indépendant en Perl dont les résultats sont stockés dans des fichiers. Cela permet de remplacer facilement un module, par exemple pour l'adapter à un autre simulateur ou pour accélérer les calculs, en utilisant un langage compilé.

Le circuit est modélisé dans une netlist Verilog. La première étape est d'analyser la netlist pour extraire la liste des bascules et leurs cônes logiques. Ensuite, un poids pour chaque bascule et chaque moment est calculé en prenant en compte divers paramètres, comme les caractéristiques du cône logique associé. Un poids élevé représente une injection de faute ayant une forte probabilité d'apparaître dans le modèle de faute considéré.

Ensuite, les fautes possibles et leurs poids sont présentés à l'utilisateur qui peut choisir la quantité d'injections à effectuer, suivant le temps dont il dispose et la précision dont il a besoin.

À partir de la liste des fautes choisies, notre outil génère un fichier de commande pour le simulateur utilisé (Cadence NCSim). Ce fichier de commandes décrit les simulations et les injections de fautes à effectuer. Comme nous ne modifions pas le circuit, la seule possibilité d'injection de faute est l'utilisation des commandes du simulateur. Cette partie de l'outil est spécifique au simulateur.

Pendant chaque simulation, le circuit à tester est manipulé par un environnement de test (benchmark) qui lui fournit les signaux d'entrée, vérifie les signaux de sortie et écrit les résultats dans des fichiers. L'environnement de test est écrit en VHDL et utilise la bibliothèque textio pour écrire les fichiers.

Un analyseur lit ces fichiers de sortie et calcule les métriques de sécurité ainsi que d'autres analyses définies par l'utilisateur et spécifiques au circuit. Les sorties peuvent être visualisées graphiquement.

3.2 Analyse du circuit

La plupart des outils réalisés précédemment permettent à l'utilisateur d'injecter des fautes en simulation sans pour autant l'orienter vers des injections préférentielles lui permettant d'optimiser ses calculs.

D'autres outils orientent l'utilisateur vers un sous-ensemble particulier d'injections possibles. Deux approches ont été proposées : sélectionner les injections qui ont une grande chance de perturber le système (pour obtenir une borne inférieure du taux de couverture) ou sélectionner des injections représentatives de l'ensemble des fautes (pour obtenir un taux de couverture proche de celui du circuit).

DEPEND [GIY97] limite le nombre de fautes en se basant sur une analyse de la charge de travail. Güthoff et Sieh [GS95] proposent d'analyser l'exécution sans fautes (*golden run*) d'instructions exécutées sur un processeur pour ne simuler que

les fautes sur les registres très utilisés en ne sélectionnant que les moments d'injection où la valeur du registre est significative.

La technique de l'extension de faute (*fault expansion*) [SJPB95] consiste à regrouper les fautes et à n'en simuler qu'un représentant. Cependant, cette méthode n'est réellement efficace que lorsque les groupes sont de grande taille par rapport au nombre de fautes total [WTSI94].

Dans notre approche, le modèle de faute est représentatif de perturbations physiques. Il doit être combiné à un modèle de circuit pour améliorer et optimiser l'analyse de l'effet des fautes.

En plus de la modélisation du circuit, PAFI peut utiliser d'autres informations comme le fichier d'analyse des délais et le résultat de l'exécution sans faute, notamment pour utiliser des informations dynamiques sur les changements de valeur des signaux.

Les informations relatives à l'exécution correcte, obtenues par simulation du circuit sans injection de faute, peuvent être utilisées pour prendre en compte les activations dynamiques des régions du circuit, en éliminant les fautes dans les régions inactives.

Ensuite, un poids est attaché à chaque bascule. Il est calculé suivant des critères structurels et/ou dynamiques. La composante structurelle est basée sur le cône logique de la bascule et prend en compte les délais, la largeur et la profondeur du cône. Le composante dynamique est basée sur les informations de l'exécution sans faute et peut changer au cours du temps. La formule exacte pour le calcul du poids dépend du modèle de faute choisi : l'utilisateur peut vouloir cibler les cônes ayant une caractéristique structurelle donnée, ou favoriser les informations sur l'exécution sans faute. Nous proposons deux pondération possibles, basées sur la fonction et sur le délai du cône logique.

Cette analyse nous fournit les triplets $bascule, moment, poids$ qui informe l'utilisateur et lui permet de choisir la quantité d'injection à effectuer, suivant le temps disponible et la précision voulue.

3.2.1 Modèles de circuit

Le choix du modèle de circuit est important dans la mesure où il influe sur les possibilités d'analyse du circuit et conditionne la pertinence des résultats fournis par l'analyse en simulation.

Plusieurs contraintes s'appliquent au choix du modèle de circuit. Il est préférable d'utiliser si possible une modélisation déjà utilisée dans l'industrie pour faciliter l'intégration de l'outil dans le flot de conception existant du partenaire industriel. D'un autre côté, le modèle doit pouvoir être assez précis pour prendre en compte le modèle de faute choisi.

Il faut donc faire des compromis pour être à la fois pertinent physiquement sans surcharger le modèle, ce qui rendrait la simulation trop lente.

3.2.1.1 Modélisation analogique

La modélisation la plus précise d'un circuit prend en compte la physique des matériaux utilisés et la technologie des transistors correspondante. Les simulations de ce type sont effectuées avec un simulateur analogique comme SPICE.

Dans ce contexte, un circuit est un ensemble de capacités, de résistances et de transistors en parallèle ou en série, suivant la fonction logique du circuit. L'ensemble des valeurs d'un signal n'est pas discret car il représente la tension appliquée dans un fil.

Ce niveau de modélisation très fin est l'un des plus proche du circuit physique mais sa simulation nécessite un grand nombre de calculs, puisqu'on doit calculer la valeur de chacun des signaux à chaque instant. Il permet notamment de calculer des délais de propagation des signaux très précis au prix de calculs très coûteux en temps, par exemple lors de la synthèse et de l'optimisation depuis un langage de haut niveau (VHDL, Verilog).

3.2.1.2 Modélisation logique hiérarchique

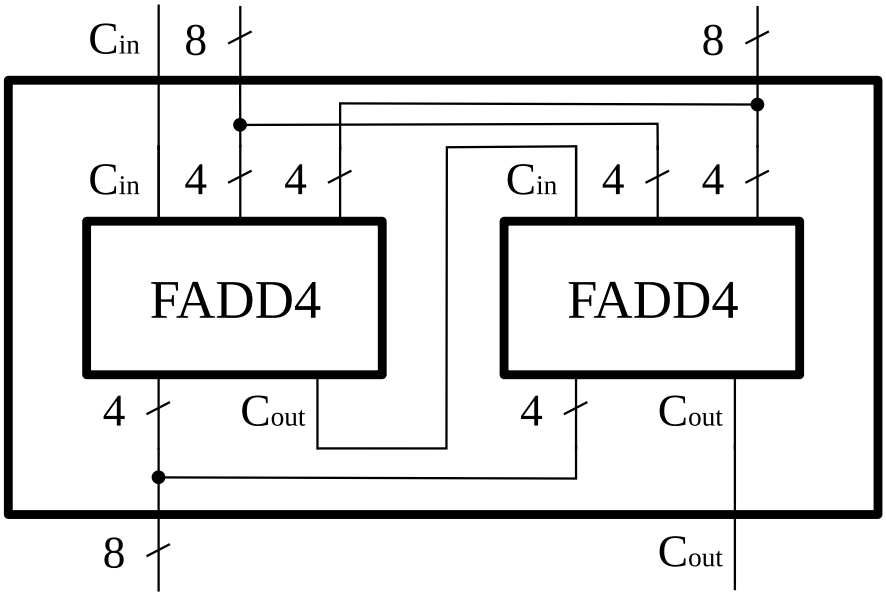

Fig. 3.3 – Exemple d'une modélisation hiérarchique

La modélisation de circuit par une hiérarchie est issue du besoin industriel de réutilisabilité, de substitution et de protection de la propriété industrielle des com-

posants. Les simulations logiques de circuits hiérarchiques utilisent des langages de haut niveau.

Un circuit est vu comme une interconnexion de composants, eux-mêmes constitués de sous-composants. Les connexions sont des signaux qui portent des valeurs discrètes (par exemple : 0, 1 ou X). La figure 3.3 présente un exemple d'additionneur 8 bits réalisé à partir de deux additionneurs 4 bits.

Ces composants sont alors réutilisables lors de la conception de circuits différents. En normalisant les interfaces d'un composant, on peut le replacer par un autre plus performant ou moins onéreux, sans avoir à changer le reste du circuit. De plus, un circuit peut être conçu et simulé avec une description fonctionnelle des composants sans avoir besoin de leur description physique précise.

Ce type de modélisation est plutôt éloigné du circuit final puisque les composants sont ensuite combinés et optimisés. La frontière entre les composants n'est plus aussi nette que dans la modélisation et certains composants sont combinés pour s'adapter à la bibliothèque de composants utilisés.

Par exemple, il est difficile de déduire des délais de propagation représentatifs puisque des optimisations automatiques n'ont pas encore été effectuées, alors qu'elles ont un impact important sur ces délais, notamment dans le but de maximiser la fréquence de fonctionnement. De plus, à ce niveau de description, la correspondance entre les composants utilisés et ceux des bibliothèques technologiques des fondeurs n'a pas encore été faite.

Ce niveau de modélisation est adapté à l'étude des conséquences de fautes sur le fonctionnement du circuit mais il n'est pas assez précis pour prendre en compte un modèle de fautes représentatif de perturbations physiques.

3.2.1.3 Modélisation logique aplatie

Le circuit est modélisé comme une interconnexion de composants de base de la bibliothèque de composants utilisés. Plus précisément, le circuit est un ensemble de points mémoire connectés par des portes combinatoires (figure 3.4).

L'état d'un circuit synchrone est stocké dans ces éléments de mémorisation. Pour faciliter l'analyse du circuit, nous avons choisi de considérer les composants mémoires comme des ensembles de points mémoire d'un seul bit (bascules).

À chaque front d'horloge, l'état de chacune de ces bascules peut changer, selon un ensemble de portes qui définit leurs valeurs à partir de l'état précédent et des entrées du circuit. L'ensemble des portes qui calcule la valeur d'une bascule définit son cône logique. Ce modèle nous permet de simuler le comportement fonctionnel du circuit tout en gardant assez de détails pour permettre de plus amples analyses comme l'estimation du délai des cônes logiques.

C'est une modélisation logique : l'ensemble des valeurs des signaux est discret,

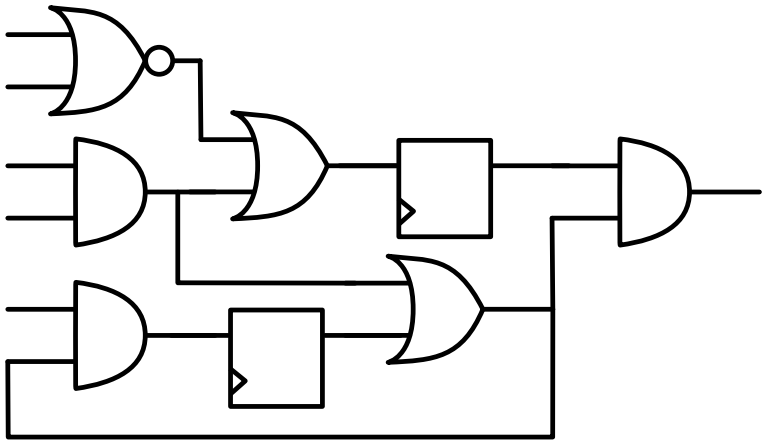

FIG. 3.4 – Exemple d'une modélisation logique aplatie

ce qui accélère la simulation par rapport à la modélisation analogique. Ce modèle permet une simulation événementielle, où la valeur d'un signal ne change pas tant qu'aucun des signaux dont elle dépend ne change. Ce type de modèle est simulable de la même manière que la modélisation hiérarchique, et utilise les mêmes langages.

Comme il n'y a pas de hiérarchie, les optimisations entre les composants de haut niveau ont été faites en amont. Le placement-routage n'a pas encore été effectué, mais on peut déjà évaluer une partie des caractéristiques importantes du circuit comme la surface et le temps critique.

Nous avons choisi cette modélisation parce qu'elle permet d'être proche du circuit final tout en restant au niveau logique, même s'il est difficile de retrouver une fonction précise dans un ensemble de portes interconnectées, surtout après optimisation.

3.2.1.4 Autres modélisations possibles

L'outil d'injection DEPEND [GIY97] (Cf. section 1.3.3) utilise une modélisation en objets C++ qui bénéficie alors des propriétés des langages objets, notamment l'héritage et la surcharge. Cette modélisation se rapproche du langage SystemC qui vise à la modélisation comportementale des circuits en utilisant des classes C++. Même si cela a l'avantage de rapprocher la conception de circuit des bonnes pratiques de la conception logicielle, les modèles obtenus ne représentent que l'aspect fonctionnel et comportemental des circuits physiques correspondants.

D'autres possibilités peuvent être envisagées pour rapprocher la modélisation de circuits d'autres domaines. Par exemple, [HZ04] utilise BHDL, une méthode ayant pour objectif de formaliser la conception sûre de circuits en utilisant des méthodes formelles (Méthode B).

Cependant, ces approches nécessitent de nouveaux outils ou des modifications

aux outils existants dans l'industrie. C'est pour cette raison que ces modélisations restent souvent expérimentales et ne sont pas encore utilisées à grande échelle.

3.2.2 Pondérations

3.2.2.1 Pondération par la fonction

Les facteurs fonctionnels sont relatifs à l'utilisation de la bascule au regard de l'ensemble du circuit : type de donnée, valeur critique pour le déroulement du circuit, etc. Ces informations doivent être fournies par l'utilisateur puisqu'elles ne peuvent pas être extraites du circuit. Nous considérons trois types de données critiques : les données secrètes, les données de contrôle et les données de sorties.

Une donnée secrète doit être isolée de l'extérieur du circuit pour ne pas révéler d'information. S'il existe une donnée observable (i.e. qui peut être lue) dont la valeur dépend d'une donnée secrète, alors cette dernière doit également être protégée. En effet, toute perturbation générant un résultat faux pourrait permettre d'extraire la donnée secrète par cryptanalyse.

Pendant le fonctionnement du circuit, les données de contrôle représentent les résultats des tests et déterminent le comportement du circuit. Une perturbation sur les bascules correspondantes peut avoir une incidence sur la sécurité, par exemple en validant une authentification fausse ou en permettant un accès à des ressources protégées.

Les données de sorties sont facilement observables. Une perturbation qui ne modifie qu'une partie du résultat facilite leur cryptanalyse. Dusart, Letourneux et Vivolo [DLV03] le démontrent sur AES en ne modifiant qu'un quart du résultat (4 octets sur les 16 du résultat).

La première étape de notre méthode de pondération consiste à affecter un poids aux bascules qui manipulent ces trois types de données. Cette connaissance des fonctions des bascules du circuit est fournie par l'utilisateur, en amont de l'analyse du circuit.

Ensuite, comme les bascules sont interconnectées les unes aux autres par des cônes logiques, on propage les poids. Si une perturbation sur une bascule B (cf. figure 3.5) change le comportement du circuit, des perturbations sur les bascules rencontrées en suivant les chemins de propagation (A_i et C_i) peuvent aussi induire une modification du fonctionnement du circuit. Les fautes produites sur ces bascules sont appelées des *perturbations liées*, puisqu'elle se produisent sur des bascules qui sont sur le même chemin de propagation. Elles se décomposent en *perturbations antérieures* et *perturbations postérieures*.

Perturbations antérieures Une faute injectée sur une des bascules d'entrée du cône logique de la bascule B peut avoir une influence sur la valeur de cette der-

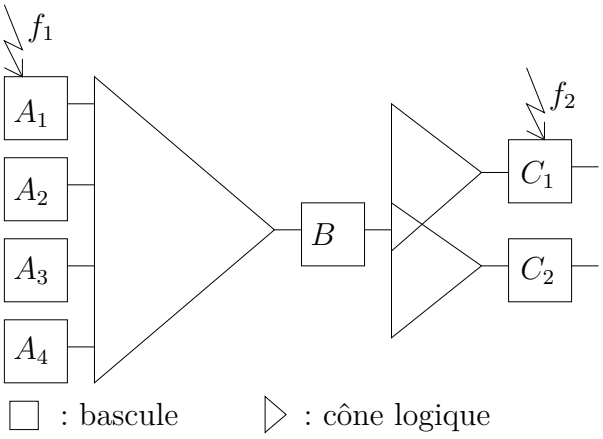

FIG. 3.5 – Perturbations liées (f_1 : antérieure, f_2 : postérieure)

nière. Ces bascules sont notées A_i et ces perturbations sont appelées *perturbations antérieures* car B est dans les chemins de propagation issus des A_i.

On observe sur le cas illustré par la figure 3.6 qu'un changement de la valeur d'une des bascules A_i est équivalent à une faute sur la bascule B.

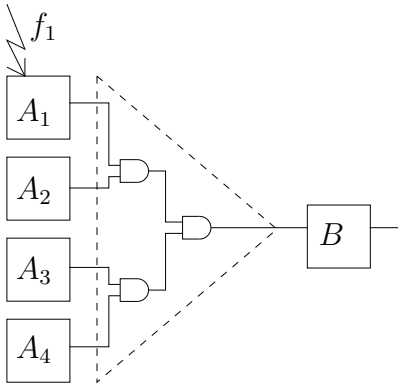

FIG. 3.6 – Exemple de perturbation antérieure

Si B manipule une donnée secrète, une faute sur l'adressage nécessaire à la récupération de celle-ci correspond à une perturbation antérieure, puisque le comportement est modifié même si la bascule contenant la donnée secrète n'est pas directement touchée par la faute.

Si B manipule une donnée de contrôle, une perturbation antérieure à celle-ci modifiera le fonctionnement du circuit si le résultat du test est altéré. Il est même possible d'obtenir un blocage du circuit, ce qui est un comportement acceptable si aucun résultat erroné ne sort du circuit.

Une faute injectée proche des sorties causera une modification partielle du ré-

sultat qui permettra une attaque par analyse différentielle. De ce fait, on considère que les données de sorties issues de B sont des données critiques compte tenu des perturbations antérieures possibles.

Ainsi, la pondération d'une bascule B peut induire un phénomène en cascade sur les poids des bascules rencontrées en remontant les chemins de propagations (A_i).

Perturbations postérieures De même, perturber les bascules de sortie des cônes logiques dont B est une des entrées va potentiellement changer l'exécution d'une partie du circuit. Ces bascules seront notées C_i et ces perturbations seront appelées *perturbations postérieures* car les C_i sont dans les chemins de propagation passant par B.

On observe sur le cas illustré par la figure 3.7 qu'un changement de la valeur d'une des bascules suivant B à le même impact sur une partie du circuit (celle dépendant de C_1) qu'une faute sur B.

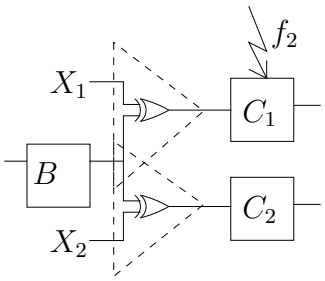

FIG. 3.7 – Exemple de perturbation postérieure

Par exemple, une bascule contenant un résultat intermédiaire dépendant d'une donnée secrète peut révéler des informations si elle est perturbée. Les travaux de Yen et Joye [YJ00] montrent que l'injection d'une *safe error* (i.e. erreur qui ne produit pas un résultat faux) sur un résultat intermédiaire lors d'une exponentiation permet de récupérer des informations secrètes.

Une perturbation postérieure sur des données de contrôle peut dérouter une partie du circuit de son exécution normale. Le circuit peut alors se retrouver dans un état imprévu, voire illégal au regard des spécifications de sécurité requises.

En prenant en compte les perturbations postérieures, une pondération sur une bascule B entraîne une pondération sur les bascules des cônes logiques dépendants de celle-ci (c'est-à-dire les C_i).

Détermination de la pondération par la fonction Les pondérations vont s'atténuer en descendant ou remontant les chemins de propagation tout en se combinant entre elles (algorithme 1), ce qui fera apparaître les points d'injection potentiellement intéressants, puisqu'ils perturberont plusieurs points critiques du circuit.

Algorithme 1 Algorithme de pondération des bascules

Entrée : $circuit \leftarrow$ description du circuit
Entrée : $nouvelle\_liste \leftarrow$ ensemble des couples $(bascule\_critique, ponderation)$
Entrée : $ancienne\_liste \leftarrow \{\}$
Sortie : la liste des pondérations finales

```
tant que nouvelle_liste ≠ ancienne_liste faire
  ancienne_liste ← nouvelle_liste
  pour chaque bascule dans circuit faire
    tmp_ponderation ← Calcul_ponderation(bascule, circuit, ancienne_liste)
    si tmp_ponderation > 0 alors
      nouvelle_liste ← nouvelle_liste ∪ {(bascule, tmp_ponderation)}
    fin si
  fin pour
fin tant que
return nouvelle_liste
```

Les attaques sur AES [Gir04][DLV03][PQ03] se basent sur des fautes proches de la sortie qui ont la propriété d'être également proches des clés de tours (secrètes) : la faute est en même temps une perturbation antérieure à la sortie et une perturbation postérieure à des données secrètes.

Cette approche a été développée dans [FFBM06, FFT+06] mais n'a pas été implémentée puisqu'elle s'écarte de l'objectif d'utiliser un modèle de faute qui soit justifiable physiquement.

3.2.2.2 Pondération par le délai

Cette pondération s'appuie sur le modèle de faute de délai décrit à la section 2.3.4. Une porte appartient au cône logique relatif à une bascule si une de ses sorties est connectée sur l'entrée de la bascule ou sur l'entrée d'une des autres portes appartenant au cône logique. Le poids d'une bascule est donc calculé à partir du délai de son cône logique.

Le modèle utilisé pour les délais est similaire à celui utilisé dans [Smi85]. Un chemin de délai entre une entrée et la sortie du cône est la somme des délais intermédiaires depuis l'entrée jusqu'à la sortie. Lorsqu'il y a plusieurs chemins possibles, le délai considéré est le maximum des délais des différents chemins puisque c'est seulement après ce délai que la propagation du signal d'entrée sera terminée. La maximum des délais de chacun des cônes logiques est le chemin critique, c'est-à-dire celui dont la propagation nécessite le plus de temps. C'est ce chemin critique qui définit la fréquence de fonctionnement du circuit.

Le délai d'une bascule peut être calculé de manière statique ou dynamique.

Le délai statique est calculé en se basant exclusivement sur la structure du circuit fourni, suivant le modèle de la section 3.2.1, ainsi que sur le fichier de délais

correspondant.

Le délai dynamique ajoute la prise en compte de l'exécution sans faute du circuit. Cela permet de calculer le délai de propagation réel lorsque seulement certaines entrées du cône logique changent d'état.

Délai statique Le poids d'une bascule est le maximum des délais des chemins entre chaque entrée du cône logique correspondant et sa sortie (Cf. figure 3.8). Ce calcul est le pire cas de propagation, c'est-à-dire qu'on considère qu'à chaque coup d'horloge, l'entrée du cône la plus éloignée (au sens des délais de propagation) change de valeur.

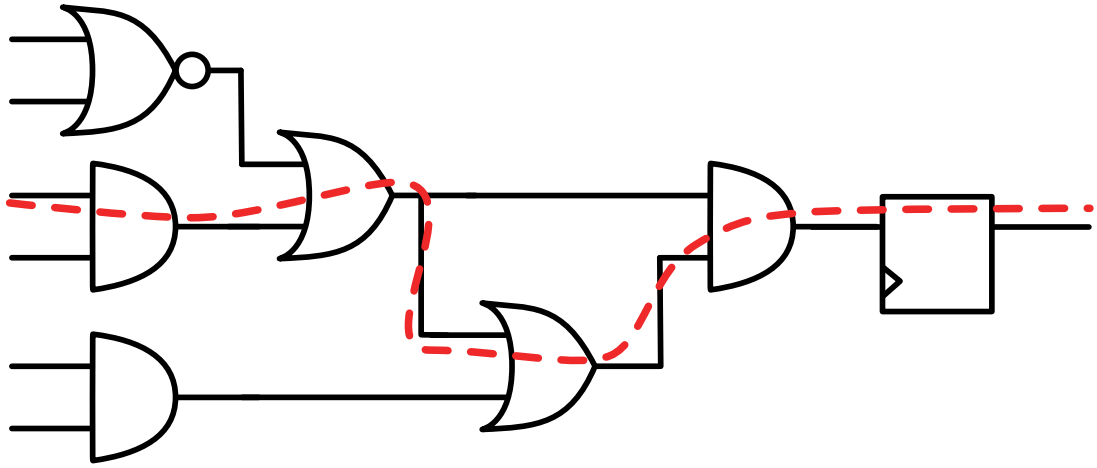

FIG. 3.8 – Délai maximum d'un cône logique

Techniquement, ce délai peut être calculé pour chaque cône logique en utilisant le fichier SDF (Standard Delay Format) correspondant. SDF est un format de représentation des délais qui a été standardisé par l'IEEE. Ce fichier fournit les délais entre chaques entrées et sorties de chacun des composants de la netlist.

À partir de la netlist, on peut obtenir le fichier SDF de ces composants internes. Dans le cas le plus courant où le circuit est d'abord modélisé hiérarchiquement, l'outil utilisé pour générer la netlist nous fournit également le fichier au format SDF correspondant, en utilisant les informations des bibliothèques de composants des fondeurs.

À l'aide de ces délais, PAFI calcule automatiquement les délais maximum des cônes logiques et les associe aux bascules. Le fichier SDF donne la liste des délais de propagation à travers les composants, entre chaque entrée et chaque sortie. L'algorithme doit calculer le délai de chaque chemin de chaque cône logique, en prenant en compte le fait qu'une entrée peut avoir plusieurs chemins jusqu'à la sortie du cône

logique. Tous les chemins doivent donc être parcourus pour trouver celui qui est le plus long, contrairement à la recherche de chemin minimum qui peut être arrêtée dès que les chemins restants à parcourir sont plus longs que le chemin minimum courant.

Cette étape prend beaucoup de temps, l'algorithme a été optimisé en prenant comme heuristique que les composants sont peu reliés entre eux : un composant non-mémorisant a souvent une ou deux sorties pour au maximum environ trois ou quatre entrées. Par exemple, pour un bit de donnée de l'AES, on a un cône de 2733 fils pour 1126 composants. Chaque interconnexion est relié à 2 entrées/sorties, sauf pour la sortie du cône et les entrées du circuit qui représentent 53 fils. Il reste 2680 interconnexions, soit 5360 entrées/sorties pour les 1126 composants. On obtient 4,76 entrées/sorties par composant en moyenne (3,76 entrées vers 1 sortie). Ceci s'explique par le fait que la netlist utilise des composants de base de la bibliothèque du fondeur qui ont un nombre d'interconnexions réduit et qui sont optimisés au maximum en taille.

Pour une interconnexion donnée, il est alors plus efficace de chercher les chemins parmi ceux qui lui sont reliés directement (par l'intermédiaire d'un composant) plutôt que parmi tous ceux vers qui il a un chemin. Le résultat est le même, le nombre d'itérations passe de *ln(profondeur)* à *profondeur*, mais le gain est réel : pour chaque fil, il est plus rapide de faire *profondeur* itérations vers en moyenne 3,76 entrées que *ln(profondeur)* itérations vers tous les fils connectés par un chemin, dont le nombre est en moyenne de 3,76 à la 1re itération, puis augmente à chaque itération.

Pour calculer le plus long chemin, l'algorithme doit recalculer les chemins à chaque itération. Notre algorithme fait plus d'itérations, mais cela est compensé par un nombre moins important de recalculs inutiles.

Comme certains calculs sont utilisés plusieurs fois, notamment dans le cas où des parties de cônes logiques sont partagées, on utilise un cache en mémoire.

Cette analyse prend 1h30 sur un processeur Sparc V9 à 1,6 GHz et consomme 185 Mo de mémoire pour le circuit d'AES (665 cônes, 1126 composants) qui est le cas d'étude de la section 4.1).

Délai dynamique Le délai dynamique est une amélioration du délai statique qui prend en compte l'exécution sans faute du circuit pour éliminer les injections de fautes dans les bascules stables.

Plutôt que de prendre comme poids le maximum des délais des entrées, on prend comme poids le maximum des délais des entrées qui ont changées au top d'horloge précédent. En effet, lorsqu'aucune des entrées ne commute, ajouter un délai ne provoque pas de faute : le délai dynamique du cône est nul. Si seule une entrée très proche de la sortie du cône est modifiée, le signal sera stable très rapidement.

Cette approche nécessite de récupérer les commutations des entrées des cônes logiques du circuit. Pour cela, la simulation sans fautes génère un fichier au format CSV (Comma-Separated Values).

PAFI fournit un outil qui analyse ce fichier et le transforme en un fichier contenant les changements de valeur du circuit. En combinant ces informations avec l'analyse du délai statique, on obtient le délai dynamique, qui fait varier le poids des bascules suivant les données, et donc suivant le temps.

L'analyse du fichier CSV est dépendante des données utilisées lors de l'exécution du circuit, contrairement à l'analyse de la netlist. Il faut environ 10 secondes pour effectuer l'analyse des données du fichier CSV, alors que le temps d'analyse du délai statique est de 1h30. Séparer les 2 analyses permet de tester le circuit avec un grand nombre de données de test sans avoir à refaire l'analyse du délai statique.

3.2.3 Réalisation technique

Pour faciliter l'utilisation industrielle de PAFI, il a été choisi d'utiliser une modélisation de circuit utilisée dans l'industrie (Cf. section 3.2.1), c'est-à-dire une netlist Verilog aplatie (sans hiérarchie) avant placement-routage.

Ce format est un fichier texte qui décrit les interfaces du circuit, donne la liste des fils internes, puis liste les composants de base utilisés avec leur type et les correspondances interconnexion-fil.

L'analyseur syntaxique et sémantique de Verilog utilisé a été écrit en Perl par Wilson Snyder et est disponible sur le CPAN[1] : il utilise le fichier de description du circuit et le transcrit en objets Perl manipulables par le programme. C'est la disponibilité de cet analyseur syntaxique qui nous a conduit à utiliser une netlist Verilog. Cela signifie que PAFI peut être étendu à d'autres langages s'il existe un analyseur syntaxique qui les supporte.

À partir de cette description, le premier traitement est de reconstituer les connexions entre composants. Pour savoir si un composant C_1 est connecté à un composant C_2, il est nécessaire de savoir s'ils ont chacun une interconnexion à un fil commun. De plus, le sens de cette interconnexion doit être connue pour savoir si c'est C_1 qui a pour entrée une valeur de sortie de C_2 ou l'inverse.

Une analyse de la structure du circuit est effectuée pour extraire les points et moments d'injections, ainsi que les informations structurelles associées. Cela inclut les délais de propagation des signaux, la forme et la taille des cônes logiques, les dépendances entre les points mémoires, etc.

Suivant l'analyse et la taille du circuit, ainsi que la multiplicité des connexions, le temps nécessaire pour parcourir les éléments du circuit peut être important. Le choix de Perl pour la portabilité, la disponibilité de l'analyseur Verilog et la facilité de prototypage implique un surcoût dans le temps d'analyse.

[1] Comprehensive Perl Archive Network : site regroupant les bibliothèques écrites en Perl

3.3 Injection

La description du circuit n'étant pas modifiée, seules les commandes du simulateur permettent de modifier le déroulement de la simulation du circuit. À partir de la liste de fautes à injecter choisie par l'utilisateur, PAFI génère le fichier de commandes pour le simulateur Cadence NCSim qui regroupent les commandes d'exécution combinées avec les commandes d'injection de fautes.

L'étape d'injection en simulation est entièrement automatisée. Le circuit est encapsulé dans un circuit de test (*testbench*) qui lui fournit les entrées, vérifie les sorties et consigne le résultat du circuit ainsi que les informations nécessaires à l'interprétation des résultats dans des fichiers qui seront utilisés pour l'analyse finale. Par exemple, ce fichier peut contenir la différence entre le résultat fauté et le résultat théorique.

Ce circuit de test est écrit en VHDL et permet d'utiliser les bibliothèques de haut-niveau (bibliothèque textio) pour l'écriture dans les fichiers (même si celles-ci sont évidemment non-synthétisables). Comme les fonctions de base de cette bibliothèque ne sont pas suffisantes, PAFI fournit un composant logger.vhd qui encapsule les fonctions de cette bibliothèque dans des fonctions adaptées à notre utilisation.

Même si ce circuit de test est écrit en VHDL, il n'est pas indépendant du simulateur dans la mesure où les fonctions d'écriture dans les fichiers ne sont pas normalisées et le rendent ainsi dépendant d'un simulateur particulier. Néanmoins, comme les accès fichiers sont encapsulés dans le composant logger.vhd, l'adaptation à une autre bibliothèque d'accès aux fichiers peut être faite rapidement.

Le circuit de test est en grande partie similaire à celui utilisé pour le test pendant la conception et est donc réutilisé depuis le circuit de test fonctionnel. Le principal travail est l'ajout et l'utilisation du composant de consignation dans les fichiers. Le choix du VHDL comme langage pour le circuit de test a justement été guidé par le fait que le fichier de test utilisé lors de la conception (en VHDL) du circuit peut être récupéré. En effet, les vecteurs de tests utilisés pour valider fonctionnellement le circuit couvrent un grand nombre de cas de fonctionnement

Cependant, comme le circuit est lui en Verilog, le simulateur doit supporter les simulations mixtes VHDL/Verilog. Techniquement, l'utilisation d'un circuit de test en Verilog permettrait de n'utiliser qu'un simulateur Verilog, mais comme le problème ne s'est pas posé (NCSim supporte la simulation mixte), la réutilisation du circuit de test était une option plus rapide à mettre en œuvre.

La simulation s'effectue sur ce circuit de test mais les commandes du simulateur injectant des fautes ciblent exclusivement le circuit à tester, qui est vu comme un composant du circuit de test.

L'utilisation d'un seul fichier de commandes plutôt qu'un fichier par injection a été choisi pour des raisons de performance. Avec un fichier par injection, chaque

injection effectue le chargement du circuit, la simulation avec injection de faute et le déchargement. En utilisant la commande « reset », qui recommence la simulation du circuit courant depuis le début, on peut utiliser un seul fichier de commandes et économiser le temps d'analyse du circuit qui est effectué au chargement.

Pour chaque injection, on utilise la commande « force » du simulateur NCSim qui permet d'imposer une valeur à un signal. Le circuit est tout d'abord simulé normalement jusqu'à quelques nanosecondes avant le top d'horloge. À ce moment-là, la valeur du fil d'entrée de la bascule, qui est aussi le fil de sortie du cône logique, est inversée et on avance la simulation de quelques nanosecondes. Au moment du front d'horloge, la mémorisation de la valeur inversée a lieu. Ensuite, le fil d'entrée de la bascule est « relâché » (commande « release ») et sa valeur est à nouveau le résultat du calcul de son cône logique. Enfin, la simulation continue et les résultats sont calculés. Cette méthode a l'avantage de correctement émuler la mémorisation d'une mauvaise valeur.

L'autre méthode possible est d'imposer la valeur de la sortie de la bascule pendant un cycle d'horloge. Tous les calculs pendant ce cycle d'horloge sont alors effectués en utilisant cette valeur erronée de la sortie de la bascule, comme si une faute avait eu lieu sur cette bascule pendant le cycle précédent. Cependant, un paramètre de la faute dépends alors d'une caractéristique d'exécution du circuit (la fréquence de fonctionnement utilisée) ce qui implique de changer les fautes lorsque l'on souhaite tester un même circuit à plusieurs vitesses d'horloge différentes. De plus, cette méthode d'injection est moins proche du phénomène de mémorisation erronée.

Ces méthodes sont correctes lorsque l'on considère que les bascules ont une seule entrée de mémorisation (et une seule sortie pour l'injection sur la sortie). Ce n'est pas toujours le cas puisque certains composants de mémorisation plus élaborés définissent leur valeur de sortie comme une fonction de plusieurs de leurs entrées. Le même problème s'applique aux sorties car il est fréquent d'utiliser des bascules qui possèdent 2 sorties : la valeur mémorisée et son inverse. Cependant, dans notre cas, le composant peut être remplacé par un ensemble de portes et une bascule simple, ce qui convient à notre modèle. Le choix d'utiliser exclusivement des bascules simples est fait lors de la synthèse du circuit.

Cette partie de PAFI est spécifique au simulateur, puisqu'il génère un fichier de commandes NCSim. Cependant, notre outil est modulaire et peut être adapté pour générer des fichiers de commandes pour d'autres simulateurs, et éviter ainsi d'être abandonné si le simulateur associé n'est plus maintenu, comme cela a été le cas pour MEFISTO [JAR+94].

3.4 Analyse des résultats

Les informations sur l'exécution du circuit doivent ensuite être agrégées et analysées pour évaluer le circuit et comprendre son comportement en présence de fautes.

Toutes les métriques définies dans la section 2.5 ne sont pas forcément calculables, dans la mesure où les changements de comportement général (par exemple une variation de consommation) ne sont pas accessibles en simulation.

En simulation, les informations disponibles sont le temps d'exécution, les valeurs des signaux (y compris ceux de sortie et de détection), les informations relatives au fonctionnement correct du circuit (comme le résultat attendu), les caractéristiques de la faute et les informations relatives aux attaques par faute connues.

Les métriques qui peuvent être calculées sont :

- le taux de fonctionnement correct
- le taux de faux positifs
- le taux de blocages
- le taux de blocages détectés
- le taux de résultats faux
- le taux de résultats faux exploitables
- le taux de résultats faux détectés

Le taux de fonctionnement correct peut être facilement calculé comme la proportion de sorties correctes obtenues. Cependant, la signification de « sortie correcte » doit être définie soigneusement.

Par exemple, un résultat mathématiquement valable mais qui arrive un cycle d'horloge plus tard peut être considéré comme correct ou non, suivant si ce retard fournit une information à l'attaquant ou si d'autres contraintes s'appliquent (ex : circuit temps réel).

Lorsque le circuit contient un système de protection, il faut prendre également en compte que le taux de résultats corrects qui ont malgré tout déclenché le système de détection.

Ce taux de détections parasites (faux positifs) devra être minimisé au maximum pour améliorer la qualité de la protection.

Le taux de blocages est défini par le nombre d'injection de faute qui ne se sont pas traduit par la fourniture d'un résultat, même faux et/ou tardif. C'est par exemple le cas lorsqu'une machine d'état se retrouve dans un état incohérent qui conduit à une boucle infinie ou à un arrêt des calculs.

Définir le blocage d'un circuit nécessite qu'il existe un mécanisme qui signale la fin du calcul et la fourniture du résultat.

Combiné à un système de protection, ce taux peut être affiné en taux de blocage

détecté. Dans un circuit protégé, une mesure de protection peut stopper les calculs et signaler l'attaque : un circuit bloqué dont la détection a été activée est considéré comme un comportement acceptable.

Le taux de résultats faux représente les cas où le circuit a effectivement fournit un résultat, mais que celui-ci est différent du résultat prévu.

Lorsque l'algorithme du circuit est connu, il est important de mesurer la proportion de ces calculs faux qui peuvent être utilisés pour effectuer des attaques par faute.

Si le circuit peut détecter les attaques, le mécanisme de détection pourra être évalué en s'intéressant aux résultats faux détectés, ainsi qu'aux résultats faux exploitables détectés.

D'un point de vue qualitatif, on peut affiner l'analyse en proposant de caractériser les sorties erronées, par exemple en comptant le nombre de bits faux.

De plus, l'analyse des sorties peut révéler le syndrome du comportement en faute du circuit, qui sera un point de départ pour des mécanismes de tolérance de faute. Par exemple, lorsque la machine d'état est attaquée et que le circuit se bloque, on pourra suggérer un mécanisme de protection spécifique protégeant la machine d'état.

Dans tous les cas, l'utilisateur peut adapter l'analyse à ses besoins et aux fonctionnalités du circuit.

Il est par exemple possible de valider certaines propriétés du circuit lorsque des fautes surviennent, comme une sortie pseudo-aléatoire d'un circuit de cryptographie (Cf. figure 3.9), ou la propriété de silence sur défaillance[2].

Un analyseur adapté lit les fichiers générés par le circuit de test et calcule le taux de tolérance ainsi que les analyses définies par l'utilisateur. Le résultat de ces calculs peut être affiché graphiquement, par exemple en utilisant Gnuplot.

3.5 Bilan : avantages et limites de PAFI

3.5.1 Adéquation à la méthodologie

Par rapport à la méthodologie, PAFI est plus générique dans la mesure où d'autres modèles que le modèle de faute de délai peuvent être testés. La seule limitation est au niveau du simulateur, qui doit pouvoir émuler la faute à injecter.

Le modèle de faute de délai a été émulé au niveau numérique, mais il est possible d'améliorer PAFI pour qu'il utilise un simulateur de plus bas niveau et une description du circuit adaptée.

[2] *fail-silence* : si le circuit fournit un résultat, alors celui-ci est correct

Le modèle de circuit défini dans la méthodologie est compatible avec la représentation en netlist Verilog. PAFI utilise une description conforme au modèle choisi dans la section 3.2.1.

Là aussi, PAFI peut être étendu vers d'autres modèles de circuit en modifiant l'analyseur de circuit et en adaptant le modèle de faute.

Cependant, toutes les métriques de sécurité définies dans la méthodologie ne peuvent être obtenues directement. Pour compléter l'analyse de PAFI, il est nécessaire d'ajouter une analyse manuelle, principalement pour la prise en compte des attaques connues, où l'utilisateur doit classer les sorties en fonction de leur exploitabilité.

Même s'il est possible d'améliorer PAFI dans ce sens, l'injection de fautes multiples n'a pas été implémenté.

Cela nécessite de rendre plus complexe l'analyse du circuit, qui prend déjà un temps considérable et ensuite de changer le format d'échange entre le module d'analyse du circuit et celui d'injection de fautes, pour prendre en compte toutes les informations d'injections multiples.

Le module d'injection devra aussi être modifié pour générer le fichier de commande correspondant pour modifier la valeur de plusieurs bits d'état du circuit simultanément.

3.5.2 Utilisation de scripts

L'utilisation de scripts s'est révélée indispensable pour accélérer l'utilisation du simulateur : il n'est pas possible d'effectuer des milliers de simulations si chacune d'elles nécessite une interaction avec l'utilisateur.

Le langage de script le plus indiqué pour la manipulation de fichier texte est sans conteste Perl. De plus, il est disponible sur la plupart des systèmes Unix, ce qui a facilité l'installation de PAFI dans le flot de conception industriel. L'interface texte nous a permis d'hériter de tous les utilitaires Unix de manipulation de fichiers. Par exemple, certaines analyses des résultats se résument à utiliser l'utilitaire grep (sélection de lignes d'un fichier).

De plus, l'intégration et la réutilisation sont facilitées puisque les scripts peuvent être appelés interactivement depuis d'autres programmes. Cela a été le cas lors de la comparaison de plusieurs circuits : un script automatisant toutes les campagnes a été élaboré.

Néanmoins, PAFI ne fournit pas d'interface graphique, ce qui implique un temps d'apprentissage pour l'utilisateur, ainsi que la nécessité d'être familiarisé avec l'utilisation d'une ligne de commande.

L'utilisation de scripts a exclu l'utilisation de l'interface PLI (Programming Language Interface). PLI permet l'appel de fonctions en C depuis Verilog. Ainsi, il aurait été possible d'appeler des fonctions d'écriture dans des fichiers et remplacer le circuit

de test en VHDL et la bibliothèque stdio par un circuit de test en Verilog et des programmes en C.

Cette solution n'a pas été choisie parce que l'utilisation du langage C nécessitait la recompilation à chaque modification et que le circuit de test n'était pas en Verilog.

Cependant, les performances de certains modules, principalement celui d'analyse du circuit auraient pu être améliorées en utilisant un langage compilé.

3.5.3 Outil générique et flexible

PAFI est conçu pour être un outil générique car il ne fait pas d'hypothèses sur la nature du circuit (cryptosystème, CPU, etc.). Il est donc possible de l'utiliser dans un cadre plus large que le modèle de circuit vu dans la partie 3.2.1, en redéfinissant également le modèle de faute de la partie 2.3.

De plus, la fonction qui extrait les informations de la description du circuit est adaptable aux modèles de fautes voulus. Il est ainsi possible de privilégier les injections selon des critères de sécurité, en sélectionnant les endroits d'injection suivant des propriétés des points d'injection [FFBM06, FFT+06] ou des critères physiques, en se basant sur les fautes observées sur des circuits réels.

Les analyses des résultats sont aussi modifiables à volonté : n'importe quel type d'analyse peut être demandé, avec pour seule contrainte que les informations nécessaires soient fournies dans les fichiers générés par le circuit de test (valeur intermédiaire, etc).

La contrepartie est qu'il est nécessaire de modifier une partie de l'outil suivant le circuit à analyser. Un circuit de test doit tout d'abord être élaboré pour fournir les vecteurs de test et générer les fichiers de sortie, en fournissant tous les résultats nécessaires aux analyses voulues.

L'analyseur des résultats doit aussi être adapté pour réaliser les analyses de l'utilisateur, suivant les fichiers de sortie du circuit de test et les analyses demandées. En effet, le format des sorties du circuit n'est pas normalisé pour rester flexible, mais cela nécessite que l'analyseur soit adapté à ce format.

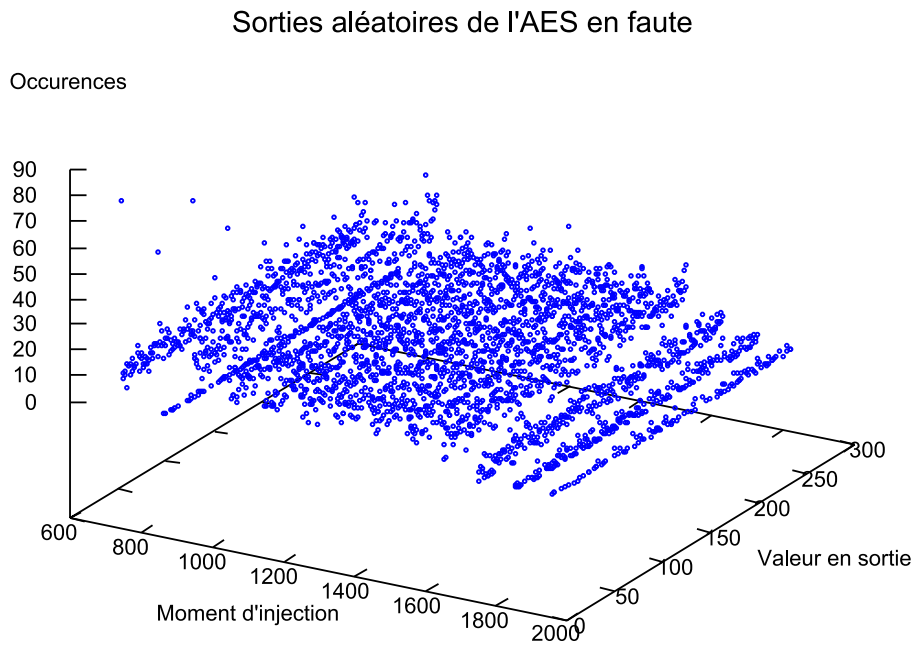

FIG. 3.9 – Visualisation des sorties de PAFI en utilisant Gnuplot

Chapitre 4

Cas d'étude : différentes implémentations de l'AES

Notre cas d'étude privilégie l'analyse des effets des fautes simples sur les circuits de cryptographie. Après avoir étudié une implémentation de l'algorithme AES, nous comparons plusieurs types d'implémentations pour étudier leur sécurité face aux fautes de délai.

Dans un premier temps, pour valider notre méthodologie et PAFI, il est nécessaire de tester la quantité et la qualité du choix des fautes. Pour cela, une injection exhaustive de fautes simples a été réalisée sur un circuit d'AES. Ensuite, l'analyse des délais a été prise en compte et les injections ont été restreintes aux bascules qui dépassent un certain seuil de délai. L'adéquation entre ces fautes et les modèles de faute nécessaires pour réaliser les attaques connues est étudiée.

Dans un deuxième temps, plusieurs implémentations différentes de l'AES ont été évaluées. Certains résultats ont été publiés dans [FM07, FF07, FTFB07]. L'objectif est de savoir si la surface et la vitesse des circuits sont liées ou non à la sécurité des circuits et de déterminer les meilleurs choix d'implémentation pour un circuit sécurisé. Pour finir, nous verrons dans quelle mesure certains circuits sont sensibles aux attaques connues en utilisant le modèle de faute en délai.

4.1 Évaluation de l'accélération de PAFI sur l'AES

AES [Sta01] est un algorithme classique de chiffrement par bloc, standardisé par le NIST pour succéder au DES. C'est un réseau de substitutions et de permutations qui prend en entrée un texte clair sur 128 bits et une clé de chiffrement de 128, 192 ou 256 bits.

4.1.1 Algorithme

Cette section présente les spécificités de l'algorithme AES. Le lecteur pourra de référer à la section 1.2 pour une vue d'ensemble de la cryptographie actuelle.

L'AES [Sta01] est le standard actuel de cryptographie symétrique par bloc du NIST. Il est plus sécurisé que le DES (clé plus longue) et plus rapide que le Triple DES en matériel et en logiciel.

C'est un réseau de substitutions et de permutations qui prend en entrée un texte clair de 128 bits et une clé de chiffrement de 128, 192 ou 256 bits.

Nous nous intéressons à la version utilisant une clé de 128 bits. Cet AES effectue 10 rondes pendant lesquelles l'entrée est mélangée en utilisant une clé de ronde calculée à partir de la clé de la ronde précédente. La première entrée est le texte clair et la clé de la première ronde est la clé de chiffrement.

4.1.1.1 Données

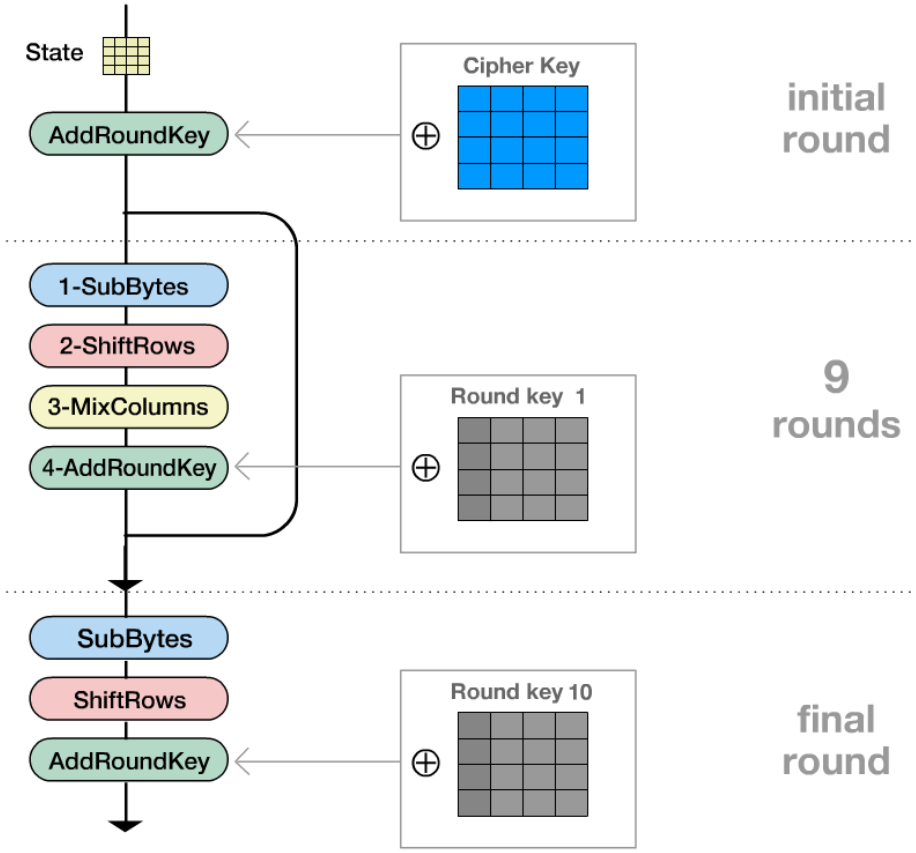

FIG. 4.1 – Diagramme du bloc de chiffrement AES [Zeb04]

Le chemin de données est une séquence de 10 rondes (figure 4.1). Les 128 bits de

données forment une matrice 4x4 de 16 octets. Une ronde normale peut être divisée en 4 étapes : SubBytes, ShiftRows, MixColumns, AddRoundKey. Avant la première ronde, un AddRoundKey entre la clé et les données est effectué. La dernière ronde ne fait pas de MixColumns.

1. SubBytes utilise une table de substitution (appelée SBox) pour transformer chaque octet. Cette substitution est équivalente à une transformation affine sur le corps de Gallois $GF(2^8)$. Cette substitution rend la ronde non-linéaire.
2. ShiftRows opère sur les lignes de 4 octets : il décale cycliquement chacune des lignes de 0, 1, 2, 3 octets respectivement. Cela déstructure les colonnes à chaque ronde en les faisant devenir des diagonales.
3. MixColumns applique une transformation linéaire sur $GF(2^8)$ sur chaque colonne (4 octets), ce qui diffuse la valeur de chacun des octets sur les 3 autres. Cette étape diffuse chaque octet seulement sur un quart des données, mais 2 MixColumns combinés à un ShiftRows rendent la diffusion complète sur toutes les données de l'AES. Cela revient à une multiplication sur $GF(2^8)$ de matrice de l'état courant S par une matrice constante C.

$$\text{C•S=} \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \bullet \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

4. AddRoundKey calcule un XOR (ou-exclusif) entre l'entrée et la clé de ronde. C'est la seule étape de la ronde qui fait intervenir la valeur de la clé.

4.1.1.2 Clé de ronde

La clé est vue comme une suite de colonne de 4 octets. Les clés de 128 bits, 192 bits et 256 bits correspondent respectivement à 4, 6 et 8 colonnes.

Le processus complet est décrit en détails dans [Sta01] et nous ne décrivons que la version utilisant une clé de 128 bits (4 colonnes).

Le calcul de la clé de ronde de l'AES à l'étape $n+1$ utilise la clé de ronde de l'étape n. La clé de la première ronde est la clé de chiffrement.

Pour chaque ronde, deux transformations (RotWord et SubBytes) sont effectuées sur la dernière colonne de la clé de ronde courante (figure 4.2).

Une constante de ronde (Rcon, définie dans [Sta01]) est ajoutée, pour prendre en compte le numéro de la ronde dans la nouvelle clé. Cette constante a pour but de modifier la clé générée suivant la ronde : une clé de ronde ne générera pas la même clé de ronde suivante à la ronde 2 ou à la ronde 5. Ainsi, une clé de ronde n'engendre par la même nouvelle clé de ronde si elle n'est pas dans le même numéro de ronde.

La nouvelle clé de ronde est obtenue colonne par colonne : la 1re colonne est le résultat du XOR entre la colonne modifiée et l'ancienne 1re colonne. Chaque

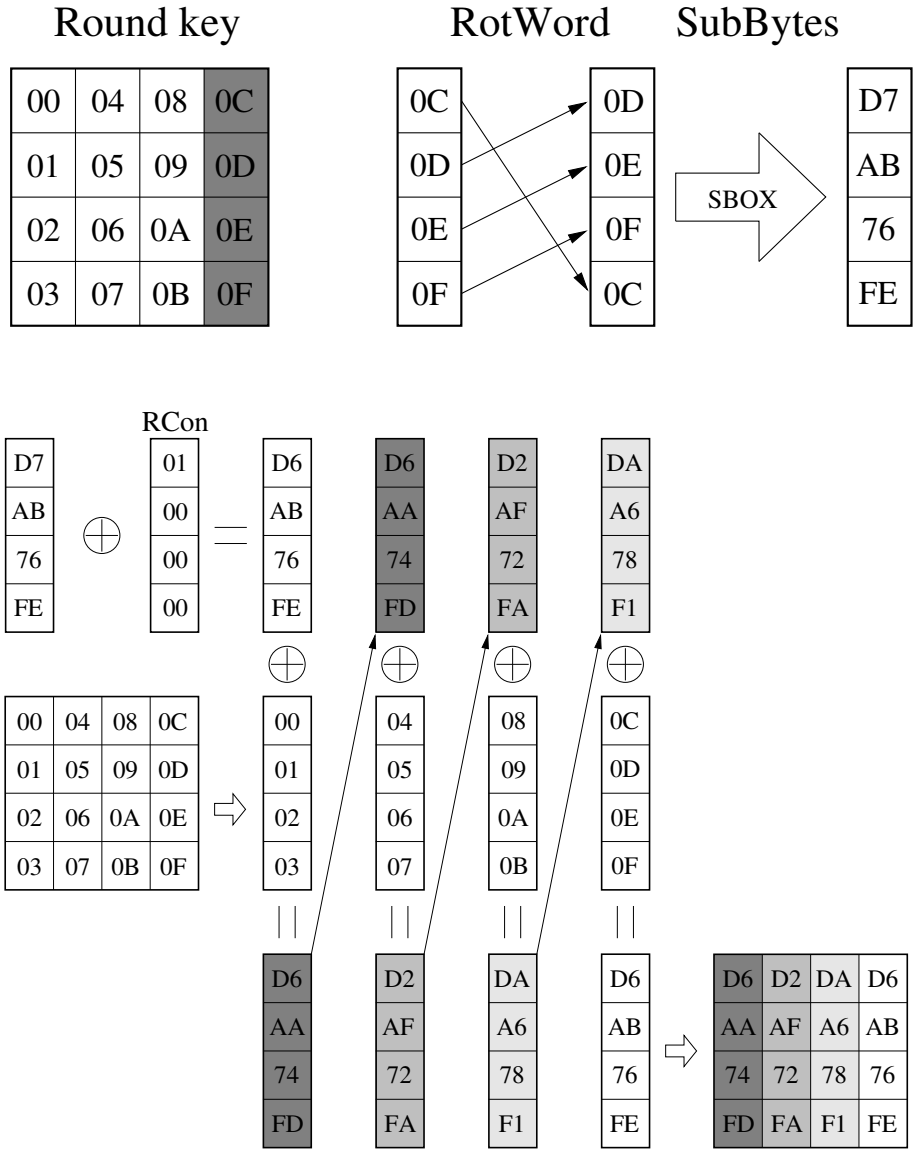

FIG. 4.2 – RotWord et SubBytes appliqués à la dernière colonne de la clé de ronde (gauche), calcul de la prochaine clé de ronde (droite)

colonne suivante est seulement l'ancienne colonne ajoutée à la colonne calculée juste précédemment. À la fin du calcul, on obtient le même nombre de nouvelles colonnes qu'en entrée.

La diffusion d'une étape de la clé est de plus de trois octets en moyenne pour une clé de 128 bits.

La clé de ronde peut être calculée à la volée (comme une donnée de ronde) ou être calculée une seule fois, stockée dans le circuit et récupérée lorsque nécessaire.

4.1.2 Implémentation

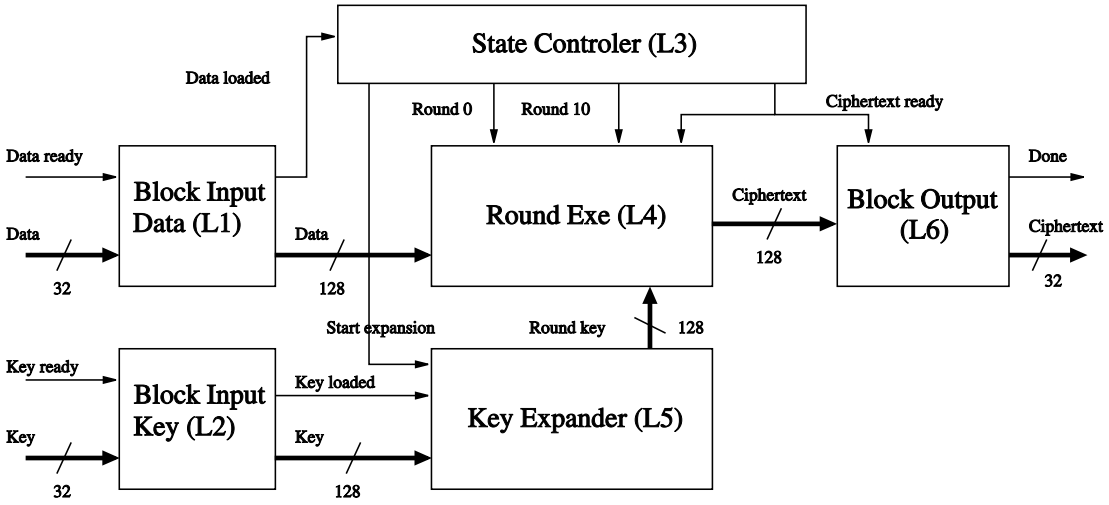

FIG. 4.3 – Architecture de l'AES

Le circuit d'AES implémenté prend en entrée 4 mots de 32 bits pour les données et 4 mots de 32 bits pour la clé de chiffrement. Une fois le chiffrement terminé, le résultat est fourni sous la forme de 4 mots de 32 bits.

L'AES est composé de 6 parties notées L1 à L6 (figure 4.3).

L1 et L2 sont des buffers d'entrée pour le texte clair et la clé de chiffrement. Les données comme la clé sont représentées sur 128 bits : les buffers permettent de les charger par mots de 32 bits.

L3 est une machine d'état qui synchronise les interactions entre les autres parties.

L4 contient les 4 étapes de la ronde de l'AES décrites à la section 4.1.1.1 et un registre de 128 bits représentant l'état courant entre les rondes.

L5 est le bloc qui calcule la clé de la ronde courante. Il contient un registre de 128 bit qui est initialisé avec la clé de chiffrement.

L6 est un buffer de sortie de 128 bits qui permet de sortir le résultat en 4 mots de 32 bits, de manière symétrique par rapport à L1 ou L2 pour les entrées.

4.1.3 Attaques par faute sur l'AES

Pour pouvoir discerner les attaques par faute qui produisent des erreurs exploitables, il faut étudier les attaques par faute connues et les modèles de faute nécessaires.

Les attaques sont classées suivant l'étendue des fautes qu'elles nécessitent.

4.1.3.1 Attaque sur 1 bit

Giraud, DFA on AES, Attaque 1 La première cryptanalyse par faute de l'AES a été publiée par Giraud [Gir04]. La faute injectée est une inversion de bit, dont la localisation est inconnue, sur les 128 bits d'entrée de l'étape de SubBytes de la dernière ronde. Cette faute est équivalente à une faute sur la clé de ronde ou sur l'entrée de l'AddRoundKey précédent.

La dernière roude ne comporte qu'un ShiftRows (SR) et un SubBytes (SB), mais pas de MixColumns (MC). La faute sur un bit d'un octet provoque une faute sur le résultat du SubBytes de cet octet. Le ShiftRows ne fait que décaler les octets, il déplace simplement l'octet, avant le XOR avec la 10^{e} clé de ronde. L'erreur sur la sortie se limite donc à un seul octet.

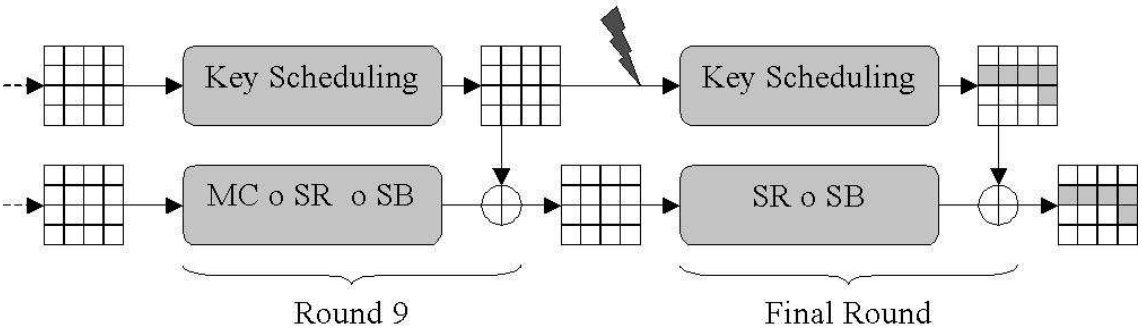

FIG. 4.4 – 1ère attaque de Giraud [Gir04]

On note $m9$ l'octet sur lequel la faute e est injectée, et k_{10} l'octet de la 10^{e} clé de ronde qui va être ajouté par XOR. Le résultat correct est : $SubBytes(m9) \oplus k_{10}$. Le résultat en injectant la faute est : $SubBytes(m9 \oplus e) \oplus k_{10}$. En ajoutant ces 2 résultats, on obtient :

$$d = SubBytes(m9) \oplus k_{10} \oplus SubBytes(m9 \oplus e) \oplus k_{10} = SubBytes(m9) \oplus SubBytes(m9 \oplus e)$$

On connaît l'octet où a été injecté e, il y a 8 possibilités pour le bit qui a été touché. Pour chaque possibilité, on fait la liste des $m9$ possibles et on réitère le processus avec d'autres e. Avec 3 fautes en moyenne, on obtient la valeur de $m9$. En ajoutant $SubBytes(m9)$ au résultat correct, on obtient l'octet k_{10} de la clé. Avec

environ 50 injections, on obtient la 10^{e} clé de ronde, qui permet de remonter à la clé de chiffrement complète.

Choukri et Tunstall, Round reduction using faults La cible de cette attaque [CT05] est la machine d'état qui synchronise les différentes parties d'un circuit d'AES. L'intérêt d'attaquer la machine d'état est de provoquer la sortie d'un résultat intermédiaire alors que le chiffrement n'est pas terminé.

Par exemple, le passage de 0 à 1 du bit de poids fort d'un compteur peut provoquer la fin du calcul et la sortie d'une valeur intermédiaire. En arrêtant le calcul avant même le début du chiffrement, il est possible de sortir la somme de la clé et du texte clair !

Blömer et Krummel, Fault Based Collision Attack on AES Dans cette attaque [BK06], on commence par chiffrer tous les messages correspondants aux 256 valeurs d'un octet $P_{0..255}$.

On chiffre ensuite un message Q qui est l'un des P_k, en provoquant une faute à la sortie de la première SBox. La sortie erronée correspond à l'un des P_i (disons, P_f), le résultat final du calcul sera donc l'un des 256 chiffrements, qui ne diffèrent que d'un seul bit par rapport au chiffrement normal de Q. En notant K la clé, on a :

$$SBox(Q \oplus K) \oplus SBox(P_f \oplus K) = 2^e$$

Des tables précalculées donnent, à partir de e et de la somme de 2 octets y, les couples (a, b) qui satisfont $a \oplus b = y$ et $SBox(a) \oplus SBox(b) = 2^e$. On connaît e puisque c'est la différence en sortie, et $y = Q \oplus K \oplus P_f \oplus K = Q \oplus P_f$, on obtient alors $Q \oplus K$ et $P_f \oplus K$.

Les sommes des tables précalculées pointent pour 129 valeurs vers une liste vide, pour 126 vers 2 couples et pour une valeur vers 4 couples, ce qui implique que dans la plupart des cas, 2 chiffrement avec injection de faute seront nécessaires.

4.1.3.2 Attaques sur un octet

Giraud, DFA on AES, Attaque 2 Giraud a aussi proposé une autre attaque plus complexe [Gir04], mais avec des fautes moins précises puisqu'elles sont limitées à un octet plutôt qu'à un bit. Cette attaque se fait en 3 étapes.

On perturbe la dernière colonne juste avant le dernier keyschedule (K_9), ce qui entraîne une différence en sortie sur 5 octets (K_{10}). Le 5^{e} octet modifié donne des informations sur la faute qui a été injectée, ce qui permet d'obtenir un octet de la dernière colonne de la clé de l'avant-dernière ronde. On répète cette étape quatre fois et on obtient les 4 octets de la dernière colonne de l'avant-dernière clé de ronde (K_9).

On faute alors la dernière colonne avant l'avant-dernier keyschedule (K_8), ce qui entraîne une différence en sortie sur 10 octets. De la même manière, un de ces octets donne des informations sur la faute propagée pendant le dernier keyschedule. De

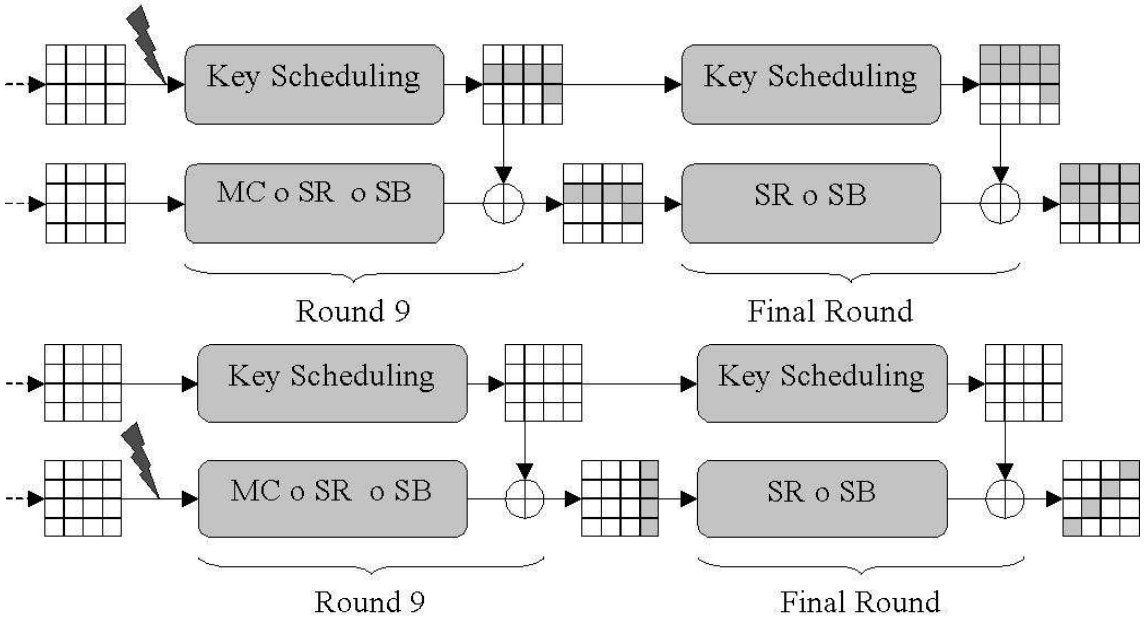

FIG. 4.5 – 2ème attaque de Giraud [Gir04]

plus, comme la dernière colonne de l'avant-dernière clé de ronde (K_9) est connue, il est possible d'obtenir 3 octets de façon sûre et 130 possibilités pour le 4ème octet de la dernière colonne de celle d'avant (K_8), qui nous donne l'avant-dernière colonne de K_9.

Dans la dernière étape, la faute est injectée sur la dernière colonne, avant la ronde 9. Tout la dernière colonne est fautée avant la ronde 10. On ne connaît pas la valeur de cette colonne, ni la faute injectée : on fait une recherche exhaustive sur les 2^{42} possibilités pour trouver celles qui satisfont les différences observées en sortie.

Comme on a la dernière colonne avant la ronde 10, et les 2 dernières colonnes de l'avant-dernière clé de ronde (K_9), on arrive à calculer de proche en proche 14 octets de la dernière clé de ronde. Les 2 derniers sont ensuite trouvés par recherche exhaustive.

Chen et Yen, Differential Fault Analysis on AES Key Schedule Chen et Yen ont proposé une attaque similaire [CY03], puisqu'elle utilise les mêmes 2 premières étapes, sauf pour l'octet incertain de la dernière colonne de K_8.

La faute est injectée sur l'avant-dernière colonne de K_8, ce qui induit 2 octets faux sur les 2 dernières colonnes de K_9. Les fautes sur K_9 se propage sur K_{10} en une ligne et un octet faux et sur M_9 par 2 octets faux sur les 2 dernières colonnes. L'octet de la dernière colonne est ajouté au même de K_{10} que lorsqu'il n'y a pas de faute : il nous permet de retrouver la faute injectée. La différence de l'octet de

l'avant-dernière colonne peut alors être retrouvée.

On obtient donc 2 octets de K_{10} et en réitérant le processus, on en obtient 8. En appliquant la même méthode que Giraud on remonte à tous les octets sauf le 0, 1 et le 4, qui sont recherchés exhaustivement. Giraud proposait une complexité en 2^{42}, ici on est en 2^{24}.

Piret et Quisquater, A differential Fault Attack Technique against AES Cette attaque [PQ03] est la plus avancée actuellement.

La faute a lieu sur un seul octet avant le dernier MixColumns (ronde 9). Cela fait 1020 possibilités de faute. Après le MixColumns, on obtient une propagation de cette différence sur toute la colonne. Cette différence n'est pas affectée par l'ajout de la clé, ni le ShiftRows. On précalcule donc la liste des 1020 possibilités de différence avant le SubBytes.

Ensuite, on utilise le texte clair et le chiffré faux correspondant, et on fait des hypothèses sur la colonne de la dernière clé de ronde. On commence par faire une hypothèse sur les 2 premiers octets de la colonne (2^{16}), et pour les candidats sélectionnés parmi les 1020 possibilités, on ajoute une hypothèse sur le 3^{e} octet, puis sur le 4^{e}.

Avec un seul couple clair/chiffré, on obtient la clé dans 97% des cas. Pour les 3% restants, on a 16 candidats.

L'attaque peut même être encore plus efficace si la faute est injectée une ronde plus tôt : toutes les colonnes ont alors un octet faux et l'analyse peut être effectuée sur chacune des colonnes séparément.

Dusart, Letourneux et Vivolo, Differential Fault Analysis on AES Cette attaque [DLV03] est du même type que la précédente mais n'utilise pas de table précalculée. Elle utilise une corrélation entre les fautes en sortie et les constantes utilisées lors des multiplications du MixColumns. Le résultat est moins précis que Piret et Quisquater et l'attaque est plus fastidieuse.

Blömer et Seifert, Fault based cryptanalysis of the Advanced Encryption Standard Ces attaques [BS03] sont dérivées du principe de « safe error » de Yen et Joye [YJ00]. La faute injectée est un collage à 0 sur un bit après l'AddRoundKey. Si l'exécution du circuit est modifiée (résultat faux, arrêt), c'est que le collage a produit une erreur et que donc le bit était à 1 dans l'exécution sans faute. Les auteurs ont aussi proposés d'autres attaques avec moins de contraintes.

La dernière attaque utilise un collage à 0 de toute l'entrée d'une SBox. Ensuite, on chiffre 256 messages en faisant varier la valeur de l'octet en entrée de cette SBox. On obtient alors une colision : lorsque l'octet est de la valeur de la clé, on obtient le même résultat que lors du chiffrement avec le collage à 0.

4.1.3.3 Attaques sur plusieurs octets d'une colonne

Moradi, Shalmani et Salmasizadeh, A Generalized Method of differential fault attack against AES cryptosystem Cette attaque [MSS06] est une généralisation des deux précédentes sur une colonne.

Il est nécessaire de savoir si on est dans le cas de 3 ou de 4 octets faux. Dans les 2 cas, on précalcule un dictionnaire des différences possibles (2^{26} et 2^{32}) avant le SubBytes. On cherche ensuite 3 ou 4 erreurs (une sur chaque octet) qui permettent d'obtenir les différences observées. Pour cela, il faut effectuer 6 injections différentes dans le cas de 3 octets faux et 1500 pour 4 octets faux.

Cette attaque pourrait être utilisée simplement en changeant les textes clairs d'entrée pour obtenir artificiellement des différences avant le SubBytes. Cependant, il n'y a aucune certitude que les différences soient sur 3 ou 4 octets à coup sûr.

Attaque	**Type de faute requis**
Giraud 1	1 bit de donnée
Choukri et Tunstall	1 bit de donnée
Blömer and Krummel	1 bit de donnée
Giraud 2	1 octet de donnée ou de clé
Chen et Yen	1 octet de clé
Piret et Quisquater	1 octet d'une colonne de donnée
Dusart, Letourneux et Vivolo	1 octet de donnée
Blömer et Seifert	1 octet de donnée
Moradi, Shalmani et Salmasizadeh	max. 3 octets d'une colonne de donnée

FIG. 4.6 – Synthèse des attaques sur l'AES

4.1.4 Hypothèses et résultats d'expérience

Dans un premier temps, nous avons réalisé une injection exhaustive comme base d'évaluation. Cela nous a permis ensuite d'estimer le gain de temps obtenu en sélectionnant les moments d'injections les plus propices à l'occurrence de fautes.

Ensuite, l'analyse des délais des bascules du circuit de l'AES a confirmé les ordres de grandeurs des délais des bascules suivant leur fonction dans l'algorithme AES, mais a également montré que les optimisations effectuées lors de la synthèse peuvent provoquer des différences de délai.

Finalement, nous avons testé l'accélération apportée par une injection de fautes restreinte aux endroits les plus sensibles aux fautes produites par ajout de délai.

4.1.4.1 Injection exhaustive sur l'AES

Nous avons commencé par une injection exhaustive de fautes simples sur les bascules de l'AES, pour avoir un étalon de départ et mesurer concrètement l'apport

de PAFI. L'implémentation de référence utilise un SubBytes sous forme de table de correspondance (LUT) et un MixColumns utilisant une addition non-conditionnelle. D'autres implémentations seront présentées dans la section 4.2.1.

Le circuit de test consiste à envoyer le texte et la clé donnée en exemple dans le standard de l'AES et à comparer le résultat (quand il y en a un) avec le résultat attendu.

En utilisant une horloge à 10 MHz, les données et la clé sont chargées à 750 ns. Le calcul a lieu entre 750 ns et 1950 ns, avec un cycle d'horloge de 100 ns. Les résultats sont attendus à 1950 ns (4 premiers octets) jusqu'à 2350 ns.

Injecter seulement sur les bascules est déjà une accélération, puisque le circuit possède plus de 14000 fils, dont la valeur aurait pu être la cible de nos injections, mais sans garantie de provoquer une erreur. Nous avons donc injecté sur les 665 bascules du circuit, pendant le calcul qui a pris 13 cycles, ce qui a nécessité 8645 simulations où des bit-flips ont été injectés.

Les résultats sont visibles sur la figure 4.7 et le tableau 4.1. En moyenne, 60.2% des fautes injectées n'ont pas eu de répercussions en sortie et 27.6% des fautes provoquent des résultats avec 16 octets faux.

Résultat	Nombre d'injections	Pourcentage
Correct	5210	60,2%
Faux	3404	39,4%
dont faux exploitables	130	1,5%
Blocage	31	0,4%
Total	8645	100%

TAB. 4.1 – Sécurité lors de l'injection exhautive sur l'AES

Les fautes injectées au début du calcul (750 et 850 ns) produisent des erreurs qui sont très dépendantes de l'implémentation de l'AES. À 750 ns, les données sont transférées des buffers d'entrée vers l'état du circuit, on obtient un taux d'erreur comparable à celui obtenu au milieu du fonctionnement. À 850 ns, le circuit initialise le calcul : seule la machine d'état a une valeur importante et toutes les autres bascules gardent leurs valeurs précédentes. La première ronde commence réellement à 950 ns.

Entre 950 ns et 1750 ns, les résultats ne comportent pas de fautes ou sont complètement faux (16 octets différents). Certaines injections proches de la fin du calcul provoquent des erreurs sur 4 octets : cela vérifie les propriétés de diffusion de l'AES (4 octets par ronde, donc sur chaque octet après 2 rondes).

Les sorties avec seulement un octet de faux sont dues à une injection juste avant la sortie du résultat. Elles correspondent en fait au résultat avec un seul bit erroné.

Comme le souligne les attaques connues ([DLV03, PQ03]), AES est particulièrement sensible aux attaques par faute qui calculent un résultat partiellement erroné.

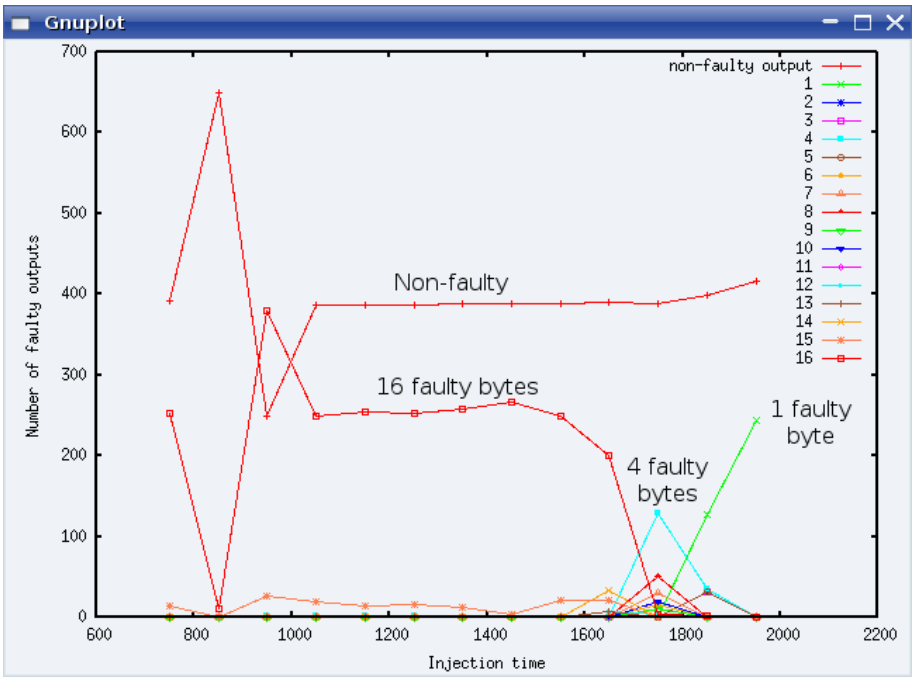

FIG. 4.7 – Faulty output bytes on AES

Sur une injection exhaustive, les fautes qui provoquent ce type de résultat représentent 1,5% des fautes injectées.

Notre analyse correspond aux résultats des injections par faute : les erreurs sur 4 octets, qui sont exploitables, apparaissent lorsque les fautes sont injectées à la 9^{e} ronde.

Le fort taux de résultats corrects s'explique par le fait qu'un grand nombre des bascules sont en fait des buffers qui sont utilisées seulement au début ou à la fin du calcul, ce qui fait qu'une injection de faute n'a que rarement un effet.

4.1.4.2 Accélération sur les moments d'injection

Un cône logique dont les entrées ne commutent pas ne change pas de valeur. L'ajout d'un délai n'a donc aucun effet. C'est le principe issu de la pondération dynamique expliquée à la section 3.2.2.2, sans utilisation de pondération spécifique quand le cône commute : soit une des entrées du cône change et son poids est le même que lors de la pondération statique, soit aucune de ses entrées ne change et son poids est nul.

Dans le cas de l'AES, 3 phases peuvent être distinguées : le chargement des données, le calcul et le déchargement du résultat. Pour chacune de ces phases, les entrées des cônes logiques peuvent changer ou non, suivant les fonctions des bascules.

La sélection des injections sur les cônes qui commutent réellement permet de

Type	Bascules	Actives au chargement	Actives lors du calcul	Actives au déchargement
Données	128	0	128	128
Buffer de sortie	132	0	0	132
Buffer d'entrée	262	262	0	0
Calcul de clé	134	0	134	0
Machine d'état	9	9	9	0
Total	665	271	271	260
Inactives		394 (59,2%)	394 (59,2%)	405 (60,9%)

TAB. 4.2 – Commutations des cônes logiques de l'AES

réduire d'environ 60% les injections à effectuer. Comme une erreur mémorisée dans une bascule est écrasée au coup d'horloge suivant, le nombre de bascules inactives du tableau 4.2 est cohérent avec la courbe des sorties correctes.

Tout ceci s'explique aussi par le fonctionnement général du circuit : les buffers sont utilisés rarement, les bascules du chiffrement et du calcul de la clé changent souvent de valeur.

4.1.4.3 Analyse des délais

Pour déterminer quelles sont les fonctions des bascules touchées prioritairement par le modèle de faute de délais de la section 2.3.4, il est nécessaire de calculer les délais des bascules du circuit. Les délais des cônes logiques sont données par la table 4.3.

Type	min.	moy.	max.
Données	10.24	11.11	11.71
Buffer de sortie	10.51	11.60	12.28
Calcul de clé	7.35	9.20	10.88
Autres	0	0.15	3.80

TAB. 4.3 – Délais des cônes logiques de l'AES (en ns)

Les bascules ayant les cônes logiques les plus longs sont les buffers de sortie (L6). Cependant, ces buffers ne sont utilisés qu'à la fin du calcul, donc seules les fautes injectées à la fin du calcul provoquent réellement une erreur. Nous avons injecté des fautes dans les bascules avec des délais supérieurs à 12 ns (18 bascules) : les erreurs en sortie était sur un seul bit.

Pendant le calcul de l'AES, ce sont les bascules des données qui ont les délais les plus longs, par rapport à celles du calcul de la clé de ronde. En effet, le calcul de

la clé utilise seulement un SubBytes et quelques XOR, alors qu'une ronde de l'AES a en plus le MixColumns. Cela signifie qu'une quantité contrôlée de délai injectée sur tout le circuit peut provoquer des fautes seulement dans la partie concernant les données courantes.

Les cônes logiques classés dans « Autres » sont les machines d'état utilisées pour synchroniser le circuit (L3) et celles des buffers d'entrée.

L'intervalle entre le maximum des délais des bascules de la clé et des délais de celles du chemin de données est de 0,83 ns (11,71-10,88). Avec une injection de délai de cette précision, il est possible d'induire des fautes seulement dans le chemin de données sans perturber le chemin de données.

4.1.4.4 Délais des octets du chemin de données

Par construction, l'AddRoundKey et le SubBytes sont les mêmes pour chacun des octets. Les ShiftRows correspondent seulement à un croisement de fils et leurs longueurs n'ont pas été prises en compte dans le calcul de délais. Si des différences doivent apparaître, ce ne peut être qu'à cause du MixColumns. Notre hypothèse est que la moyenne de chaque colonne sera constante.

Données AES	colonne 1	colonne 2	colonne 3	colonne 4
ligne 1	*10.82*	*10.82*	*10.85*	11.13
ligne 2	11.15	11.14	11.17	**11.50**
ligne 3	11.14	11.13	11.13	11.12
ligne 4	11.07	11.07	11.07	**11.41**
moyenne	11.05	11.04	11.04	**11.29**

TAB. 4.4 – Moyenne des délais sur les octets des données de l'AES

Cependant, le tableau 4.4 montre une différence importante sur la dernière colonne, qui comporte les 2 octets de plus gros délai. De plus, sauf pour la dernière colonne, la première ligne a un plus petit délai que les autres.

Cela est dû aux optimisations réalisées lors de la synthèse, qui cherchent à réduire la surface et à maximiser l'utilisation de composants complexes de la bibliothèque. Les délais moyens de chacun des octets sont difficiles à prévoir et dépendent de beaucoup de facteurs. Cela est confirmé par les résultats de la section 4.2.5.6, où plusieurs implémentations de l'AES donnent des résultats très différents pour le délai moyen de leurs octets.

4.1.4.5 Accélération sur les endroits d'injections sensibles au délai

Pour tester les délais sur le chemin de données, nous avons sélectionné les injections sur les bascules ayant un délai de plus de 11 ns. Les bascules sélectionnées appartiennent au chemin de données et aux buffers de sortie, comme cela est visible

dans le tableau 4.3. 195 bascules ont été sélectionnées : 77 bits sont dans le chemin de données (L5) et 118 dans les buffers de sortie (L6). Ce sont donc 2535 fautes qui ont été injectées.

FIG. 4.8 – Octets fautés en sortie de l'AES (délai > 11 ns)

Les résultats sont visibles sur la figure 4.8. Les fautes dans les buffers de sortie produisent les 118 résultats corrects jusqu'à 1850 ns. Après 1850 ns, les résultats comportent une seule faute, puisque leur valeurs ne se propagent pas.

Les autres erreurs en sortie sont la conséquence des fautes injectées dans le chemin de données. Ainsi, toutes les erreurs observées avant 1850 ns sont le résultat d'une injection dans le chemin de données.

Résultat	Nombre d'injections	Pourcentage
Correct	1579	62,3%
Faux	956	37,7%
dont faux exploitables	77	3%
Blocage	0	0%
Total	2535	100%

TAB. 4.5 – Sécurité de l'AES lors de l'injection sur les bascules de délai supérieur à 11 ns

Comme le montre, le tableau 4.5, une plus grande proportion des erreurs sont sur 4 octets et peuvent donc être exploitées en utilisant des attaques connues [DLV03, PQ03].

Notre méthodologie, qui utilise un modèle de faute de délai, permet de réduire significativement (de 8645 à 2535, -70%) le nombre de fautes à injecter sur le circuit tout en doublant le nombre de cas où la sortie erronée peut être exploitée.

Le nombre de blocage est nul, puisque les délais des machines d'état ne sont pas touchées car leur délai est très faible. Cette propriété est importante, notamment dans un contexte expérimental, où il sera possible d'automatiser la détection de la fin du calcul du circuit.

4.2 Effet des délais sur la sécurité de plusieurs implémentations de l'AES

Le principal intérêt de PAFI est de comparer la sécurité de circuits aux fonctionnalités similaires. L'étude a été effectuée sur différentes implémentations d'un circuit d'AES. On connaissait le principe du compromis temps/surface qui veut qu'en optimisant la surface d'un circuit, on obtient un débit plus réduit. Dans ce travail, l'apport principal est l'impact du facteur sécurité sur les contraintes de temps et de surface sur un AES non-sécurisé.

4.2.1 Différentes implémentations de l'AES

Dans toutes les implémentations suivantes, les données sont mémorisées après l'AddRoundKey dans le chemin de données. Dans le processus de calcul de la clé de ronde, la mémorisation est effectuée avant le calcul. ShiftRows est une permutation cyclique des octets donc son implémentation n'est qu'un croisement de fils et AddRoundKey est un XOR bit-à-bit. Ainsi, il semble difficile d'optimiser les descriptions physiques de ces transformations. En conséquence, les choix d'optimisation n'ont d'effet que sur les implémentations de SubBytes et MixColumns.

La figure 4.9 présente le fonctionnement simplifié de l'AES. Les cadres gris représentent les éléments mémorisants. Le multiplexeur 1 permet de choisir les données d'entrée au premier tour et le résultat de la ronde précédente ensuite. Les données sont ensuite ajoutées par XOR à la clé de ronde, puis mémorisées. Le multiplexeur 2 a la même fonction que le multiplexeur 1 mais sur la clé de chiffrement : la clé en entrée est utilisée au début, puis les clés de ronde sont utilisées. Le 3^{e} multiplexeur n'est utilisé qu'à la fin du calcul pour effectuer la dernière ronde sans le MixColumns.

4.2.2 Différents SubBytes

Le SubBytes est utilisé à la fois dans le chemin de données et dans le calcul de de la clé de ronde de l'AES. 3 implémentations différentes ont été testées :

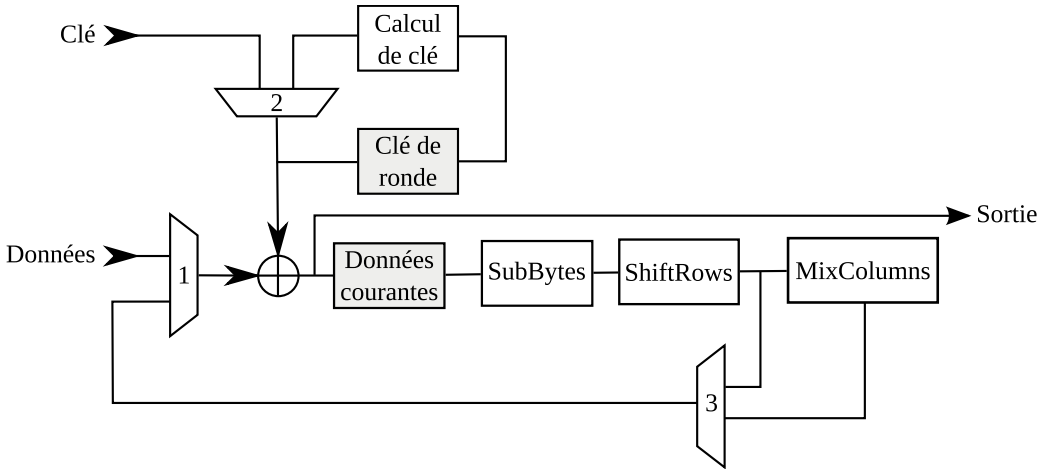

FIG. 4.9 – Fonctionnement simplifié de l'AES

- Look-Up Table (LUT) : une table de correspondance est utilisée
- « arbre et-ou » : chaque feuille est un résultat du SubBytes
- Opérations dans $GF(2^4)$: on calcule dans $GF(2^4)$ plutôt que dans $GF(2^8)$

4.2.2.1 Look-Up Table (LUT)

Cette implémentation donne directement le résultat du calcul de la SBox à partir de la valeur utilisée en entrée. Dans cette version, l'octet de l'entrée de la SBox est représenté par xy, où x et y sont respectivement les 4 bits de poids fort et de poids faible. L'octet de sortie est à la colonne x et à la colonne y de la table 4.10 définie dans le standard [Sta01].

4.2.2.2 Arbre et-ou

Cette structure est divisée en 2 parties.

La première partie calcule les 256 combinaisons possibles depuis les 8 bits d'entrée de la SBox grâce à des portes logiques ET (notées '.'). Soit $b_7b_6b_5b_4b_3b_2b_1b_0$, un octet d'entrée de la SBox, et V_j, un signal à 1 si et seulement si l'octet d'entrée représente j (compris entre 0 à 255). Ensuite, en utilisant des portes ET, on calcule V_{255}, V_{254}, ..., V_0 (respectivement $b_7 . b_6 . b_5 . b_4 . b_3 . b_2 . b_1 . b_0$, $b_7 . b_6 . b_5 . b_4 . b_3 . b_2 . b_1 . \overline{b_0}$, ..., $\overline{b_7} . \overline{b_6} . \overline{b_5} . \overline{b_4} . \overline{b_3} . \overline{b_2} . \overline{b_1} . \overline{b_0}$).

Soit $SB_V(i)$ l'ensemble des valeurs possibles en entrée des SBox telles que le bit i de la sortie est égal à 1. La deuxième partie calcule un OU des 128 valeurs V_j des ensembles $SB_V(i)$. Ainsi, si un des éléments V_j est égal à 1, la sortie sera aussi égale à 1.

On obtient le comportement désiré : lorsqu'une valeur est fournie en entrée, le V_j correspondant est à 1 et les sorties correspondantes seront à 1.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

FIG. 4.10 – Table de correspondance du SubBytes de l'AES [Sta01]

4.2.2.3 Opérations dans $GF(2^4)$

Dans cette structure, présentée dans [WOL02], un élément $a \in GF(2^8)$ est représenté par un polynôme avec des coefficients dans $GF(2^4)$, $a \cong a_h.x + a_l$, $a \in GF(2^8)$, a_h, $a_l \in GF(2^4)$ et sera décrit par la paire $[a_h, a_l]$. Chacun de ces coefficients est représenté sur 4 bits.

L'addition de 2 polynômes est réalisée par addition indépendante de chacun des coefficients. La multiplication et l'inversion de 2 polynômes nécessitent une étape de réduction modulaire pour que le résultat soit un polynôme également. Le polynôme irréductible utilisé est $n(x) = x^2 + x + e$.

$m_4(x) = x^4 + x + 1$ est le polynôme utilisé dans les multiplications dans $GF(2^4)$.

La structure résultante est donnée par la figure 4.11. Sur cette figure, le composant « map » fait la transformation dans $GF(2^4)$. « $\oplus$ » représente une addition, « $^{\wedge} - 1$ » une inversion, et « $\otimes$ » une multiplication dans $GF(2^4)$. Ces 2 dernières opérations doivent être calculées en prenant en compte $m_4(x)$. « $^{\wedge}2$» et « $\otimes e$» sont 2 cas particuliers du composant de multiplication dans $GF(2^4)$. « map^{-1}» effectue la transformations inverse du map, pour revenir dans $GF(2^8)$. Ce type de transformation peut aussi être appliqué au MixColumns.

4.2.3 Différents MixColumns

Dans le chiffrement AES, les multiplications dans $GF(2^8)$ sont calculées avec 3 valeurs différentes : '01', '02' et '03'. Un octet, nommé b, peut être représenté

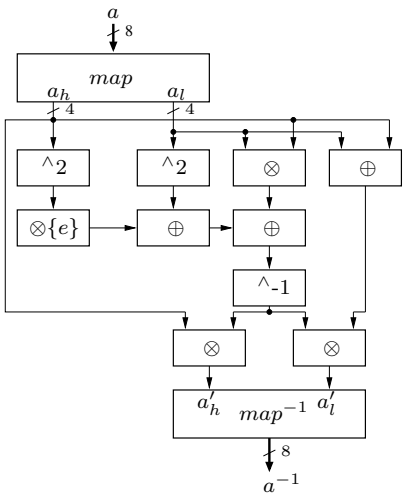

FIG. 4.11 – SBox utilisant des opérateurs dans $GF(2^4)$

sous forme polynomiale : $b(x) = b_7.x^7 + b_6.x^6 + b_5.x^5 + b_4.x^4 + b_3.x^3 + b_2.x^2 + b_1.x + b_0$. 4 structures différentes pour les multiplications vont être présentées : l'une d'elles utilise des opérations dans $GF(2^4)$, deux sont basées sur l'addition, conditionnelle ou non, du polynôme $m(x)$ et la dernière calcule directement chaque bit de la multiplication par réduction classique.

4.2.3.1 Opérations dans $GF(2^4)$

Dans cette structure, 2 éléments a et $b \in GF(2^8)$ (défini par le polynôme irréductible $m(x) = x^8 + x^4 + x^3 + x + 1$) sont représentés comme des polynômes ayant des coefficients dans $GF(2^4)$, $a \cong a_h + a_l$, (respectivement $b \cong b_h + b_l$), $a \in GF(2^8)$ ($b \in GF(2^8)$), $a_h, a_l \in GF(2^4)$ ($b_h, b_l \in GF(2^4)$) et sont notés par les paires $[a_h, a_l]$ and $[b_h, b_l]$. Si 2 octets d'entrée a et b sont multipliés dans cette représentation, une multiplication dans $GF(2^8)$ peut être accomplie en 3 multiplications, 4 additions et une multiplication par une constante, toutes dans $GF(2^4)$ [SU96].

La structure résultante est montrée par la figure 4.12. Elle utilise les mêmes composants de base que la SBox basée sur des opérations dans $GF(2^4)$.

4.2.3.2 Addition conditionnelle

Multiplier un octet par '01' préserve sa valeur.

Multiplier un octet par '02' est équivalent à multiplier l'octet dans sa forme polynomiale par x : $x.b(x) = b_7.x^8 + b_6.x^7 + b_5.x^6 + b_4.x^5 + b_3.x^4 + b_2.x^3 + b_1.x^2 + b_0.x$. Le résultat de cette multiplication peut être obtenu en le réduisant modulo $m(x) = x^8 + x^4 + x^3 + x + 1 = 1\{1b\}$ en hexadécimal. Si $b_7 = 0$, le résultat est déjà dans as forme réduite. Si $b_7 = 1$, la réduction et effectuée en soustrayant le polynôme

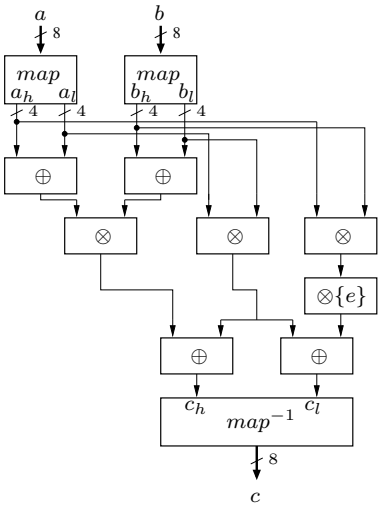

FIG. 4.12 – MixColumns utilisant des opérateurs dans $GF(2^4)$

$m(x)$. Cela implique que la multiplication par '02' peut être implémentée au niveau de l'octet comme un décalage à gauche et une addition conditionnelle par XOR avec '1b'. Cette opération sur les octets est notée xtime() dans le standard AES.

Une multiplication par '03' peut être calculée en utilisant un composant xtime() :

$$03.b(x) = (02 \oplus 01).b(x) = 02.b(x) \oplus 01.b(x) = 02.b(x) \oplus b(x)$$

4.2.3.3 Addition non-conditionnelle

Dans la version précédente, le composant xtime() calcule la multiplication par addition conditionnelle avec '1b'. Celle-ci peut être évitée. La multiplication par '02' peut être substituée par une somme de 2 valeurs intermédiaires nommée $mix2$ et $carr$.

$mix2$ décale à gauche l'octet entrant et ajoute des $'0'$: $mix2 = b_6b_5b_4b_3b_2b_1b_00$.

$carr$ est égale à '1b' si b_7 est égal à 1, 0 sinon : $carr = 000b_7b_70b_7b_7$.

Ainsi, la multiplication de l'octet par '02' peut être faite en XORant ces 2 valeurs : $02.b = mix2 \oplus carr = b_6b_5b_4b_3b_2b_1b_00 \oplus 000b_7b_70b_7b_7$.

La multiplication par '03' est donc faite par : $03.b = 02.b \oplus b = (mix2 \oplus carr) \oplus b$.

4.2.3.4 Réduction classique

Cette version du multiplieur utilise les entrées sous forme polynomiale. Si 2 opérandes a et b sont multipliées comme des polynômes $a(x)$ et $b(x)$, cela donne :

$$a(x) \bullet b(x) = (a_7.x^7 + ... + a_0.x^0) \bullet (b_7.x^7 + ... + b_0.x^0) = a_7.b_7.x^{14} + ... + a_0.b_0 \quad (4.1)$$

Les termes de degré supérieur ou égal à 8 doivent être réduit en utilisant $m(x)$: $x^8 = x.x^7 = x^4 + x^3 + x + 1$, ..., $x^{14} = x^7 + x^4 + x^3 + x$.

Si les différentes puissances de x sont remplacées dans l'équation 4.1, on obtient :

$$a(x) \bullet b(x) = a_7.b_7.(x^7 + x^4 + x^3 + x) + ... + a_0.b_0.x^0 = K_7.x^7 + ... + K_0 = \sum_{i=0}^{7}(K_i.x^i)$$

De cette manière,chaque bit de la multiplication K_i peut être calculé séparément.

4.2.4 Surface des différentes implémentations

Les implémentations correspondantes à ces SubBytes et MixColumns ont été synthétisées. La bibliothèque de portes de base pour ASIC utilisée est la « C35 CORELIB » d'AMS. Chaque synthèse a été réalisée en utilisant Synopsys Design Vision, avec une contrainte « medium-effort ». La surface totale est la somme des surfaces combinatoires, non-combinatoires et d'interconnexions. Les résultats sont dans le tableau 4.6.

SubBytes	MixColumns	Surface totale (mm^2)	Délai critique (ns)
Look-up table (LUT)	Addition conditionnelle	1281	12.77
	Addition non-conditionnelle	1288	13.21
	Réduction classique	1284	13.68
	Opérations dans $GF(2^4)$	1397	18.45
Arbre ET-OU	Addition conditionnelle	1220	13.75
	Addition non-conditionnelle	1224	13.30
	Réduction classique	1215	13.01
	Opérations dans $GF(2^4)$	1341	18.71
Opérations dans $GF(2^4)$	Addition conditionnelle	810	23.17
	Addition non-conditionnelle	804	23.12
	Réduction classique	803	22.85
	Opérations dans $GF(2^4)$	916	28.27

TAB. 4.6 – Surface des différentes implémentations d'AES

Ce tableau montre que toutes les structures qui implémentent le SubBytes en utilisant des opérations dans $GF(2^4)$ sont significativement plus petites, parce qu'elles n'ont besoin que d'environ 3000 portes logiques contre 8200 pour le SubBytes utilisant un arbre « et-ou ».

Cependant, pour chaque structure de SubBytes, l'implémentation du MixColumns utilisant les opérations dans $GF(2^4)$ augmente la surface totale du circuit. Il utilise environ 6100 portes logiques. Les autres structures de MixColumns ont besoin de moins de portes : par exemple, les structures utilisant la réduction classique du MixColumns ont besoin d'environ 4900 portes logiques.

De plus, les implémentations des composants basées sur les opérations dans $GF(2^4)$ augmentent le délai critique.

4.2.5 Hypothèses et résultats d'expérience

4.2.5.1 Précision temporelle nécessaire des fautes de délai

Dans l'AES, le chemin de données a besoin théoriquement de plus d'opérations que le calcul de la clé de ronde. Ainsi, les délais des bits du chemin de données sont plus grands que ceux du calcul de la clé de ronde.

Si un délai est ajouté au circuit entier, ce sont les bits du chemin de données qui vont être perturbés en priorité. Si le délai est suffisant, ceux du calcul de la clé le seront aussi.

Dans ce cas, il n'est pas possible de ne perturber que certains bits du calcul de la clé de ronde, ce qui restreint les attaques possibles. Seules les attaques sur le chemin de données peuvent être considérées. Par exemple, la deuxième attaque de Giraud (section 4.1.3.1) et celle de Chen et Yen (section 4.1.3.2) nécessitent des injections de fautes dans le KeySchedule seulement et ne sont donc pas possibles avec ce type de fautes.

Pour perturber seulement les bits du chemin de données, nous avons besoin d'une certaine précision dans le délai ajouté. Pour connaître la précision nécessaire, la différence entre le maximum des délais du datapath et le maximum des délais du calcul de la clé de ronde a été calculée. Cette différence est appelée « fenêtre d'attaque » dans le tableau 4.7.

SubBytes	MixColumns	Fenêtre d'attaque (ns)
Look-up table (LUT)	Addition conditionnelle	1.76
Arbre et-ou		1.91
Opérations dans $GF(2^4)$		2.39
Look-up table (LUT)	Addition non-conditionnelle	1.39
Arbre et-ou		1.59
Opérations dans $GF(2^4)$		1.65
Look-up table (LUT)	Réduction classique	0.88
Arbre et-ou		1.08
Opérations dans $GF(2^4)$		2.15
Look-up table (LUT)	Opérations dans $GF(2^4)$	7.39
Arbre et-ou		7.00
Opérations dans $GF(2^4)$		7.43

TAB. 4.7 – Fenêtres d'attaques de différentes implémentations d'AES

Ces valeurs ne donnent pas la possibilité d'injecter une faute sur un bit choisi. Elles ne représentent que la précision minimum requise pour injecter au moins une faute sur le chemin de données.

Le chemin de données utilise MixColumns alors que ce n'est pas le cas pour le calcul de la clé de ronde : c'est pour cela que les fenêtres d'attaque sont usuellement comparable pour un MixColumns donné.

Par exemple, le MixColumns avec des opérations dans $GF(2^4)$ sont plus lentes et le délai ajouté sur le chemin de données est plus grand. Un délai ajouté sur ces circuit, même imprécis dans la durée, pourra toucher plus facilement seulement les bits du chemin de données.

Cependant, pour le MixColumns en réduction classique, les fenêtres ne sont pas similaires. L'outil qui a fait la synthèse a optimisé le circuit pour réutiliser des calculs du SubBytes pour le MixColumns parce que nous avons synthétisé une netlist aplatie. L'implémentation avec les LUT ajoute seulement 0,88 ns au chemin de données alors que l'implémentation avec $GF(2^4)$ ajoute 2,15 ns.

Ainsi, cette fenêtre montre les délais ajoutés par le MixColumns mais prennent en compte les « compatibilités d'optimisation » entre deux implémentations.

4.2.5.2 Moyenne des délais des colonnes de la clé de ronde

Sur le calcul de clé de ronde, une transformation est effectuée sur la 4^{e} colonne pour calculer la nouvelle 1re colonne (schéma 4.2). Ensuite, la 2^{e} colonne est calculée par un XOR entre la nouvelle 1re colonne et l'ancienne 2^{e} colonne et la 3^{e} colonne est un XOR entre la nouvelle 2^{e} colonne et l'ancienne 3^{e}. Finalement, la nouvelle 4^{e} colonne est un XOR entre la nouvelle 3^{e} et l'ancienne 4^{e}. Si le délai de l'opération de XOR est la même pour chaque opération, l'expérience devrait mettre en évidence une augmentation linéaire des délais depuis la 1re jusqu'à la 4^{e} colonne.

SubBytes	MixColumns	Col 0	Col 1	Col 2	Col 3
Look-up table (LUT)	Addition conditionnelle	7.92	8.79	9.49	10.08
	Addition non-conditionnelle	8.93	9.77	10.55	11.11
	Réduction classique	9.03	9.87	10.70	11.24
	Opérations dans $GF(2^4)$	7.68	8.47	9.30	9.85
Arbre et-ou	Addition conditionnelle	8.64	9.50	10.33	10.88
	Addition non-conditionnelle	8.35	9.20	10.02	10.56
	Réduction classique	8.93	9.74	10.64	11.15
	Opérations dans $GF(2^4)$	8.65	9.44	10.32	10.85
Opérations dans $GF(2^4)$	Addition conditionnelle	17.49	18.30	19.10	19.65
	Addition non-conditionnelle	18.36	19.11	20.02	20.53
	Réduction classique	16.74	17.51	18.46	18.95
	Opérations dans $GF(2^4)$	17.68	18.49	19.41	19.86

Tab. 4.8 – Délais moyens sur les colonnes du calcul de clé de ronde (en ns)

Les différences entre les délais des colonnes n et $n+1$ est d'environ une porte (environ 0,5 à 0,9 ns) mais ces différences ne sont pas constantes, l'augmentation n'est pas linéaire dans le tableau 4.8. Cela est dû au fait que le XOR entre les colonnes n'est pas toujours fait avec le même type de portes.

4.2.5.3 Différences entre les bits sur le chemin de données

Sur chaque octet, les différences entre les moyennes des délais des bits sont dues aux différences entre les opérations de la SBox. Les résultats sont visibles dans le tableau 4.9.

Sub-Bytes	Mix-Columns	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
LUT	Add. cond.	11.81	12.04	11.33	12.08	12.20	11.46	11.29	11.52
	Add. non-cond.	12.43	12.28	12.00	12.40	12.42	11.97	12.40	12.37
	Réduction	12.41	12.81	12.07	12.75	12.87	12.31	12.23	12.62
	$GF(2^4)$	**15.57**	16.76	16.95	16.39	**17.35**	16.85	16.63	**17.35**
Et-ou	Add. cond.	12.17	12.23	11.70	**12.47**	12.96	**11.60**	**11.47**	11.90
	Add. non-cond.	12.35	12.17	12.14	**12.67**	12.33	**11.56**	**11.53**	11.71
	Réduction	12.09	12.31	12.58	**12.57**	12.39	**11.50**	**11.71**	12.03
	$GF(2^4)$	**16.04**	16.79	17.26	17.11	**17.57**	17.19	16.93	**17.49**
$GF(2^4)$	Add. cond.	21.82	21.95	21.53	22.17	22.14	21.58	21.67	22.21
	Add. non-cond.	22.58	22.68	21.93	22.68	22.66	22.47	22.39	22.57
	Réduction	22.03	22.45	21.57	22.11	22.26	22.03	21.88	22.42
	$GF(2^4)$	**25.52**	26.59	26.87	26.97	**27.20**	26.96	26.56	**27.23**

TAB. 4.9 – Délais moyens des bits de chaque octet du chemin de données (en ns)

Pour le SubBytes avec l'arbre « et-ou » (à l'exception du MixColumns $GF(2^4)$), le délai du bit 3 est souvent plus grand et les délais des bits 5 et 6 sont plus petits. C'est une conséquence de l'asymétrie de sa structure : le bit 3 est déterminé par plus d'entrées et nécessite donc des signaux qui transitent par plus de portes (alors que les bits 5 et 6 sont déterminés par moins d'entrées). Cela indique que l'arbre « et-ou » pour calculer le bit 3 est plus profond et que l'expression binaire utilise plus de variables.

Pour le MixColumns utilisant des opérations dans $GF(2^4)$, le délai réduit du bit 0 et le délai important des bits 4 et 7 sont la conséquence de la structure vue sur la figure 4.12. Le chemin critique du MixColumns utilisant $GF(2^4)$ traverse deux multiplications et une addition, pendant que le chemin le plus rapide traverse seulement une multiplication et une addition. Combinées aux transformations *map*, les différences de chemin conduisent aux différences de délai observées.

4.2.5.4 Différences de délai sur les bits

Certaines attaques sur le chemin de données ont besoin de fautes qui engendrent au plus un bit faux par octets avant le dernier SubBytes [Gir04].

Pour chaque implémentation, nous avons calculés le maximum des délais de chaque bit dans le but de voir si certains bits sont plus sensibles. Les résultats sont donnés par le tableau 4.10, la dernière colonne est la différence entre les 2 plus grands maximum.

SB	MC	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Diff.
LUT	Cond.	**12.54**	12.53	11.98	12.32	**12.77**	11.71	11.80	11.94	**0.23**
	Non-cond.	12.87	12.77	12.65	12.78	12.82	12.68	**13.17**	**13.21**	0.03
	Réduc.	13.13	13.19	13.17	13.20	**13.35**	12.74	**13.68**	13.20	**0.33**
	$GF(2^4)$	16.43	17.45	17.48	16.98	**18.11**	17.79	17.34	**18.45**	**0.34**
ET-OU	Cond.	12.68	12.67	12.45	**12.85**	**13.75**	11.98	12.27	12.72	**0.90**
	Non-cond.	13.07	12.88	12.84	**13.30**	**13.25**	12.03	12.03	12.89	0.05
	Réduc.	12.26	12.63	**12.95**	**13.01**	12.82	12.05	12.18	12.22	0.06
	$GF(2^4)$	17.05	18.14	18.09	18.11	**18.27**	18.10	17.67	**18.71**	**0.44**
$GF(2^4)$	Cond.	22.60	22.71	22.06	22.93	**23.17**	21.96	21.99	**23.08**	0.09
	Non-cond.	22.84	**23.12**	22.29	**23.12**	**23.12**	23.03	23.00	**23.12**	0.00
	Réduc.	22.48	22.72	21.82	22.77	22.78	**22.85**	**22.85**	22.66	0.00
	$GF(2^4)$	26.88	27.62	27.68	28.13	**28.27**	27.98	27.39	**28.16**	0.11

TAB. 4.10 – Délais maximum des bits de chaque octet du chemin de donnée (en ns)

Quand un bit a un plus grand délai maximum que les autres, il est possible d'ajouter un délai qui ne perturbera au plus que ce bit sur chaque octet.

En considérant les attaques utilisant ce modèle de faute, le SubBytes le plus sûr est celui utilisant les opérations dans $GF(2^4)$ et le MixColumns le plus sûr utilise une addition non-conditionnelle. Le SubBytes avec des LUT est presque toujours assez sensible, alors que celui avec l'arbre « et-ou » peut être sensible ou non, suivant le MixColumns.

Cette analyse doit être effectuée par octet pour savoir si chacun d'eux peut être attaqué de cette manière. Dans tous les cas, la taille du secret n'est plus de 128 bits.

[BK06] nécessite aussi un seul bit erroné, mais situé à la sortie du SubBytes. Les structures testées ne permettent pas à un attaquant d'obtenir des fautes de délai qui satisfont ce modèle.

4.2.5.5 Différences de délai sur les octets

À cause de la structure de l'AES, certaines attaques utilisent des fautes sur un ou plusieurs octets. L'attaque décrite dans [PQ03] nécessite un seul octet modifié par colonne pour récupérer les octets de clé correspondant. Une autre attaque [MSS06] utilise des fautes sur les octets d'une colonne et a besoin d'une perturbation sur au maximum 3 d'entre eux.

Pour les circuits avec un MixColumns utilisant des opérations dans $GF(2^4)$, le résultat sont visibles dans le tableau 4.11. Les 2 dernières lignes présentent les différences entre les 2 plus grand délai d'une colonne (attaque de [PQ03]) et les différences entre le plus grand et le plus petit (attaque de [MSS06]).

LUT			
16.92	16.80	16.81	16.52
17.83	18.07	18.07	17.64
18.04	18.45	18.44	16.68
17.29	17.44	17.43	17.62
Max1-Max2, Max-min			
0.21	0.39	0.37	0.02
1.12	**1.65**	**1.63**	**1.12**

Arbre et-ou			
18.23	17.36	17.36	16.98
17.62	18.02	18.02	17.17
17.59	18.11	18.11	17.37
18.71	17.96	17.93	16.27
Max1-Max2, Max-min			
0.47	0.09	0.09	0.21
1.11	**0.75**	**0.75**	**1.10**

$GF(2^4)$			
26.50	26.10	26.10	26.98
27.37	27.69	27.69	27.27
28.27	28.15	28.15	26.77
28.16	27.50	27.49	27.90
Max1-Max2, Max-min			
0.12	**0.46**	**0.46**	**0.63**
1.77	**2.05**	**2.05**	**1.13**

TAB. 4.11 – Délais maximum des octets du chemin de donnée pour le MixColumns utilisant des opérations dans $GF(2^4)$ (en ns) et différences relatives aux attaques connues

Le MixColumns avec les opérations dans $GF(2^4)$ est le plus sensible aux attaques sur les octets en utilisant des fautes de délai. Les sensibilités des autres MixColumns (non montrés) sont similaires, à part le MixColumns avec addition non-conditionnelle qui est un peu moins sensible.

Certains SubBytes se comportent bien pour un MixColumns donné, mais il n'y a pas de généralités sur la sensibilité des SubBytes.

4.2.5.6 Différences des délais moyens sur les octets

L'analyse des délais moyens de chacun des octets pour les différentes implémentations de l'AES montre de grandes différences entre les circuits (voir les résultats en annexe B). Les octets de plus grand délai moyen et de plus petit délai moyen ne sont jamais situés aux mêmes endroits sur toutes les implémentations. L'analyse par SubBytes ou par MixColumns ne montre pas de tendance sur les répartition des délais. Cela prouve que les optimisations effectuées lors de la synthèse changent beaucoup d'un circuit à un autre, même si la fonction du circuit reste la même.

La sensibilité aux fautes de délai des différentes bascules d'un circuit est donc quelque chose de peu prévisible avant synthèse du circuit. Il est alors nécessaire

d'analyser chaque circuit pour savoir quels sont les octets les plus sensibles. Cela valide le niveau de précision de la modélisation utilisée dans notre approche.

Les problèmes de différences des délais entre les différentes parties du circuit peuvent être résolus en ajoutant des délais pour avoir des temps de propagation équivalents dans tous les cônes logiques. Cela peut être utilisé pour allonger les délais du calcul de la clé de ronde et les rendre égaux à ceux du chemin de données. Dans ces conditions, un délai aura des répercutions sur un grand nombre des bascules du circuit et l'exploitation des erreurs en sortie sera plus difficile.

4.3 Bilan

Dans ce chapitre, l'outil PAFI a d'abord été utilisé pour estimer la pertinence de notre analyse et l'apport de notre approche pour la sécurité des systèmes. L'AES a été choisi comme circuit de cryptographie à tester puisque c'est un standard et qu'il est utilisé dans l'industrie. Une injection exhaustive de fautes simples a d'abord été effectuée comme référence. Ensuite, deux possibilités d'accélération ont été testées. La première prend en compte le modèle de faute de délai et sélectionne les endroits d'injection en utilisant le délai du cône logique associé. La deuxième utilise la stabilité des cônes logiques lorsque leurs entrées ne commutent pas pour éliminer les injections à ces moments-là. De plus, des hypothèses sur l'équilibrage des délais des octets ont été testées.

La deuxième partie a comparé la sécurité de différentes implémentations de l'AES face aux attaques en délais. Cette analyse a montré que le modèle de faute de délai, utilisé sur certaines implémentations, correspond aux modèles nécessaires à des attaques en faute connues.

Le SubBytes proposé dans [WOL02] qui utilise des opérations dans $GF(2^4)$ est à la fois le plus petit mais aussi le plus robuste face aux fautes de délai au niveau bit comme au niveau octet. Les 2 autres implémentations ont environ les mêmes nombres de portes. Celle utilisant une LUT est moins sensible face aux attaques utilisant des fautes sur les octets alors que les performances de l'arbre « et-ou » dépendent du MixColumns associé. Contrairement au SubBytes, l'implémentation du MixColumns qui utilisent des opérations dans $GF(2^4)$ augmente la surface du circuit mais aussi sa vulnérabilité aux attaques par faute, notamment au niveau des octets. Le cas le plus évident étant le MixColumns en $GF(2^4)$: un ajout de délai avec une précision d'environ 2 ns permet de reproduire l'attaque décrite dans [MSS06]. La sécurité des autres implémentations est comparable, même si la vulnérabilité du MixColumns utilisant une addition non-conditionnelle semble être un peu plus faible.

De plus, les attaques basées sur une faute sur un bit peuvent être utilisées : tout dépends de la précision de l'ajout du délai. Dans ce cas, les concepteurs de circuits sécuritaires doivent prendre en compte que ce modèle de faute est physiquement pertinent.

Une possibilité de protection est l'ajout de délais aléatoires pour que les endroits impactés par le délai soit aussi aléatoires [CHY05] ou d'égaliser au maximum les différents chemins [MS02] pour que la précision de délai nécessaire soit importante.

Conclusion

Les attaques contre les systèmes microélectroniques sont de plus en plus sophistiquées : de nouvelles méthodologies de conception industrielle doivent être considérées. Faces aux attaques par faute, les solutions actuelles sont principalement empiriques. L'approche présentée contribue à déceler au plus tôt les problèmes de conception de manière semi-automatique.

Dans le premier chapitre, l'état de l'art a été présenté. La sécurité des systèmes à ressources limitées, comme la carte à puce, est une contrainte supplémentaire dans la conception microélectronique. Les attaques sur les systèmes cryptographiques utilisent toutes les sources d'informations possibles pour retrouver les informations secrètes enfouies. L'équilibre entre sécurité et rentabilité industrielle est difficile à maintenir, surtout lorsque les circuits implémentent un algorithme très connu et que de nouvelles attaques apparaissent, comme récemment les attaques en faute. Cet état de l'art a été terminé par une présentation des modèles de faute usuels et des principaux outils d'injection de fautes.

Le second chapitre a exposé les objectifs de la thèse : fournir une méthodologie d'évaluation de la sécurité et un outil d'accélération d'injection en simulation utilisant des modèles de faute réalistes.

Après une présentation des phénomènes physiques justifiant les modèles de faute retenus, nous avons proposé des métriques adaptables pour mesurer la sécurité des circuits, avec ou sans protections.

Le troisième chapitre est consacré aux qualités et limites de l'outil PAFI, qui implémente notre méthodologie. Le modèle de circuit logique aplatie a été choisi, notamment parce qu'il apporte un compromis satisfaisant entre précision et rapidité de simulation. PAFI s'insère dans le flot de conception existant en facilitant l'analyse du circuit, l'injection de fautes et l'analyse des résultats.

Les résultats de notre approche sur le circuit de cryptographie AES ont été présentés dans le quatrième chapitre. L'approche implémentée dans PAFI a d'abord été validée en se basant sur une injection exhaustive de fautes simples comme étalon.

Ensuite, les accélérations obtenues ont été mesurées en sélectionnant les endroits d'injection grâce au modèle de faute ainsi que les moments d'injection en éliminant

les cônes logiques stables pour obtenir une diminution d'environ 40% d'injections.

Cela a permis de valider l'objectif d'accélération des injections en simulation. Enfin, la robustesse de plusieurs implémentations de l'AES soumises à des fautes de délai global a été évaluée en utilisant PAFI, ce qui atteste de la validité de la méthodologie pour l'aide à l'évaluation de la sécurité des circuits, tout au moins pour le modèle de faute choisi.

Plusieurs perspectives intéressantes sont ouvertes à la suite de ce travail.

Le modèle de faute de délai peut être étendu en considérant toutes les fautes multiples possibles parmi les bascules qui dépassent un certain seuil. Les méthodes d'accélérations sur les endroits et les moments d'injection présentées dans cette thèse devraient permettre de réduire le nombre d'injection. Une autre possibilité serait de définir une probabilité d'occurrence de faute pour les bascules dépassant le seuil.

Même si le modèle de faute en délai a été justifié, aucune publication, à notre connaissance, ne propose de quantification des délais qui peuvent être ajoutés à un circuit en perturbant son alimentation. La réalisation physique d'une des implémentations de l'AES et une validation expérimentale de ce modèle n'a pas été possible dans le temps imparti à la thèse mais fait partie des perspectives possibles pour ce travail.

De la même manière que nous avons procédé pour les différentes implémentations de l'AES, une évaluation des contre-mesures usuelles de l'AES peut être effectuée. De plus, les vulnérabilités observées peuvent donner lieu à des contremesures originales. Par exemple, une contre mesure pour l'AES serait de ne pas utiliser les mêmes implémentations des SBox pour le calcul de la clé et le chemin de donnée, de façon à égaliser les délais.

D'un point de vue technique, PAFI gagnerait en efficacité si le temps d'analyse des délais était plus rapide. Cela peut être effectué en remplaçant le composant d'analyse du circuit en Perl, par un composant dans un langage compilé performant (C, C++ ou OCaml). Néanmoins, cela nécessite un analyseur syntaxique de netlist Verilog dans le langage choisi.

Une ouverture plus large sur d'autres circuits de cryptographie, comme le DES, le RSA et les algorithmes à base de courbes elliptiques, voire des circuits de chiffrement par flot, peut être envisagée dans la mesure où PAFI ne fait pas d'hypothèses sur la fonction du circuit.

Pour conclure, ce travail de thèse s'est situé à l'intersection de 3 domaines : l'électronique, la cryptographie et l'informatique.

Il a été nécessaire de se familiariser avec les notions d'électronique et de conception des circuits, voire de physique des composants pour certains modèles de faute. Ce domaine était le moins bien connu au début de la thèse, c'est donc tout naturellement celui dans lequel j'ai le plus appris.

Au contraire, l'informatique était le domaine le mieux connu. Cela a permis d'amener toute l'expertise possible pour adapter les possibilités technologiques aux

besoins des autres domaines.

La cryptographie est un domaine aux limites de l'informatique et des mathématiques. Il était nécessaire de comprendre à la fois d'où venait la force des algorithmes de chiffrement et en même temps comment fonctionnait les attaques qui les compromettent.

C'est la diversité des domaines qui a fait la richesse de cette thèse.

Annexe A

ISO/IEC 7816

L'ISO/IEC 7816 est la famille de standards qui concernent les cartes à puces à contact.

L'ISO/IEC 7816-1 définit les caractéristiques physiques de la carte comme sa tolérance la torsion et à différents types de rayonnements, ainsi que sa taille (85.60 x 54 x 0.76 mm). La taille du circuit embarqué est au maximum 25 mm^2. Ces dimensions proviennent du standard ISO 7810 au sujet de la taille des cartes utilisées pour l'identification.

L'ISO/IEC 7816-2 spécifie les dimensions et l'emplacement des 8 contacts de la puce. Si on regarde une carte bancaire, on peut remarquer que l'emplacement de la puce a été choisie pour ne pas rentrer en conflit avec les chiffres en relief de la face avant et la bande magnétique de la face arrière. Ce standard, tout comme le précédent, est destiné aux constructeurs de cartes.

L'ISO/IEC 7816-3 standardise les signaux électriques et les protocoles de transmission. C'est ce standard qui fixe les tensions d'alimentation ainsi que le format de la réponse au reset (Answer To Reset, ATR) de la carte et qui couvre les aspects de communication au niveau physique. Ces informations sont particulièrement utiles pour les fondeurs de circuits pour cartes à puce.

L'ISO/IEC 7816-4 est le standard qui définit les commandes intersectorielles pour les échanges, c'est-à-dire les commandes APDU (Application Protocol Data Unit), la structure et l'organisation des fichiers, les codes de retour ainsi que des mécanismes de sécurité. C'est le standard le plus important pour les développeurs d'applications se basant sur le cartes à puces.

L'ISO/IEC 7816-5 définit le système de numérotation et les procédures d'enregistrement d'identificateurs d'applications. Les applications chargées dans la carte reçoivent un identifiant unique (AID) calculé de façon standardisée.

L'ISO/IEC 7816-6 spécifie les éléments de données intersectoriels, c'est-à-dire le format des objets de données (Data Objects, DO) et les méthodes d'accès. C'est ce standard qui permet par exemple à une application tierce d'accéder aux données GSM sur une carte SIM.

L'ISO/IEC 7816-7 standardise un langage de requête, le SCQL (basé sur SQL)

qui permet d'interroger une carte à puce d'une manière similaire à l'interrogation d'une base de donnée.

L'ISO/IEC 7816-8 décrit les commandes utilisées pour augmenter la sécurité des cartes à puce. C'est dans ce document que sont définis des mécanismes de cryptographie comme les signatures, les hachages et le chiffrement/déchiffrement.

L'ISO/IEC 7816-9 définit les commandes de gestion de la carte. C'est notamment dans ce standard que sont définis les commandes d'accès aux fichiers, de mise à jour sécurisé des données de la carte (par exemple, des applets) en utilisant des messages sécurisés et en vérifiant les droits d'accès.

L'ISO/IEC 7816-10 est le standard qui spécifie les signaux électriques et les réponses au reset pour les cartes synchrones.

L'ISO/IEC 7816-11 traite de l'identification par l'utilisation de la biométrie en utilisant les commandes définies dans le standard 7816-4.

L'ISO/IEC 7816-12 décrit l'interface électrique et les procédures de fonctionnement d'une interface USB sur carte à puce. Ce standard contient, entre autres, la description des spécificités des circuits intégrés comportant une interface USB, la description de la couche de donnée et de celle de contrôle.

L'ISO/IEC 7816-15 standardise une application d'information cryptographique. Cette application permet de connaitre les capacités cryptographiques de la carte, d'utiliser des mécanismes d'identifications, ainsi que plusieurs algorithme de cryptographie.

Annexe B

Délais moyens par octet des implémentations d'AES

ligne 0	12.02	12.10	12.15	12.15
ligne 1	11.80	12.09	12.10	12.06
ligne 2	12.16	12.16	12.17	11.66
ligne 3	12.28	12.21	12.21	11.70

Tab. B.1 – Moyenne des délais sur les octets des données de l'AES (ET_OU-ADD_COND)

ligne 0	11.85	12.47	12.46	12.03
ligne 1	11.98	11.92	11.94	11.64
ligne 2	11.73	12.31	12.31	11.86
ligne 3	11.78	12.37	12.36	11.89

Tab. B.2 – Moyenne des délais sur les octets des données de l'AES (ET_OU-ADD_NON_COND)

ligne 0	17.19	16.74	16.74	16.61
ligne 1	17.01	17.39	17.39	16.29
ligne 2	17.01	17.69	17.69	16.60
ligne 3	18.01	17.24	17.23	15.92

Tab. B.3 – Moyenne des délais sur les octets des données de l'AES (ET_OU-GF_16)

ligne 0	12.03	12.12	12.12	12.18
ligne 1	12.22	12.24	12.30	12.00
ligne 2	12.28	12.15	12.27	12.04
ligne 3	12.30	12.08	12.09	11.94

TAB. B.4 – Moyenne des délais sur les octets des données de l'AES (ET_OU-REDUC)

ligne 0	22.00	22.15	22.15	21.08
ligne 1	22.49	22.52	22.52	21.03
ligne 2	22.12	22.14	22.15	20.93
ligne 3	21.77	21.97	21.97	21.16

TAB. B.5 – Moyenne des délais sur les octets des données de l'AES (GF_16-ADD_COND)

ligne 0	22.55	22.65	22.65	21.91
ligne 1	22.53	22.62	22.63	22.13
ligne 2	22.59	22.74	22.74	21.91
ligne 3	22.57	22.81	22.80	22.09

TAB. B.6 – Moyenne des délais sur les octets des données de l'AES (GF_16-ADD_NON_COND)

ligne 0	25.88	25.68	25.68	26.12
ligne 1	26.71	26.98	26.99	26.63
ligne 2	27.58	27.46	27.46	26.04
ligne 3	27.45	26.98	26.98	27.19

TAB. B.7 – Moyenne des délais sur les octets des données de l'AES (GF_16-GF_16)

ligne 0	22.12	22.13	22.13	21.60
ligne 1	22.16	22.16	22.16	21.82
ligne 2	22.34	22.36	22.36	21.74
ligne 3	22.25	22.24	22.24	21.68

TAB. B.8 – Moyenne des délais sur les octets des données de l'AES (GF_16-REDUC)

ligne 0	11.78	11.76	11.76	11.52
ligne 1	11.64	11.70	11.70	11.54
ligne 2	11.82	11.92	11.92	11.67
ligne 3	11.84	11.69	11.69	11.51

TAB. B.9 – Moyenne des délais sur les octets des données de l'AES (LUT-ADD_COND)

ligne 0	12.35	12.38	12.39	12.08
ligne 1	12.37	12.32	12.31	12.20
ligne 2	12.49	12.24	12.24	12.16
ligne 3	12.40	12.18	12.24	12.18

TAB. B.10 – Moyenne des délais sur les octets des données de l'AES (LUT-ADD_NON_COND)

ligne 0	16.14	16.22	16.22	15.99
ligne 1	16.99	16.99	16.99	16.60
ligne 2	17.31	17.47	17.46	16.03
ligne 3	16.73	16.85	16.86	16.86

TAB. B.11 – Moyenne des délais sur les octets des données de l'AES (LUT-GF_16)

ligne 0	12.44	12.47	12.67	12.88
ligne 1	12.35	12.29	12.29	11.95
ligne 2	12.59	12.62	12.62	12.41
ligne 3	12.88	12.60	12.59	12.48

TAB. B.12 – Moyenne des délais sur les octets des données de l'AES (LUT-REDUC)

Bibliographie

[AAA+90] Jean Arlat, Martine Aguera, Louis Amat, Yves Crouzet, Jean-Charles Fabre, Jean-Claude Laprie, Eliane Martins, and David Powell. Fault injection for dependability validation : A methodology and some applications. *IEEE Trans. Softw. Eng.*, 16(2) :166–182, 1990.

[AFRS03] J. Arlat, J.C. Fabre, M. Rodríguez, and F. Salles. MAFALDA : A Series of Prototype Tools for the Assessment of Real Time COTS Microkernel-Based Systems. *Fault injection techniques and tools for embedded systems reliability evaluation*, pages 141–156, 2003.

[Ale96] Aleph One. Smashing the stack for fun and profit. *Phrack*, 7(49), November 1996.

[ALRL04] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1) :11–33, 2004.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. *Lecture Notes in Computer Science*, 1233 :37–51, 1997.

[BECN+04] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. Cryptology ePrint Archive, Report 2004/100, 2004.

[BK06] Johannes Blömer and Volker Krummel. Fault based collision attacks on AES. In *FDTC*, volume 4236 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2006.

[BPRR98] A. Benso, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda. EXFI : a low-cost fault injection system for embedded microprocessor-based boards. *ACM Transactions on Design Automation of Electronic Systems.*, 3(4) :626–634, 1998.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[BS03] J. Blomer and J.P. Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). *Seventh International Financial Cryptography Conference (FC 2003)*, 2003.

[CHY05] D.J. Kinniment C.A. Hoggins, C. D'Alessandro and A. Yakovlev. Securing On-Chip Operations against Timing Attacks. *Technical Report Series, NCL-EECE-MSD-TR-2005-108*, 2005.

[CI92] G. S. Choi and R. K. Iyer. FOCUS : An experimental environment for fault sensitivity analysis. *IEEE Trans. Comput.*, 41(12) :1515–1526, 1992.

[CMS98] Joao Carreira, Henrique Madeira, and Joao Gabriel Silva. Xception : A technique for the experimental evaluation of dependability in modern computers. *Software Engineering*, 24(2) :125–136, 1998.

[CNV+00] P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M. Sonza Reorda, and M. Violante. Experimentally evaluating an automatic approach for generating safety-critical software with respect to transient errors. *Nuclear Science, IEEE Transactions on*, 47(6 Part 3) :2231–2236, 2000.

[CT05] H. Choukri and M. Tunstall. Round reduction using faults. In *2nd International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'05), Edinburgh, Scotland, September*, pages 13–24, 2005.

[CY03] Chien-Ning Chen and Sung-Ming Yen. Differential fault analysis on AES key schedule and some countermeasures. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy – ACISP 2003*, volume 2727 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2003.

[DLV03] P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on a.e.s. Cryptology ePrint Archive, Report 2003/010, 2003. http ://eprint.iacr.org/.

[ELB06] Reouven ELBAZ. *Hardware Mechanisms for Secured Processor-Memory Transactions in Embedded Systems*. PhD thesis, Université Montpellier 2, 12 2006.

[EMZ+03] A. R. Ejlali, G. Miremadi, H. R. Zarandi, G. Asadi, and S. B. Sarmadi. A hybrid fault injection approach based on simulation and emulation co-operation. In *DSN-2003 : Proceedings of the 2003 International Conference on Dependable Systems and Networks*, pages 479–488, Washington, DC, USA, 2003. IEEE Computer Society.

[FF07] Olivier Faurax and Julien Francq. Security of several AES implementations against delay faults. In *Proceedings of the 12th Nordic Workshop on Secure IT Systems (NordSec 2007)*, October 2007.

[FFBM06] Olivier Faurax, Laurent Freund, Frédéric Bancel, and Traian Muntean. Une méthode générique pour l'injection de fautes dans les circuits. In

Actes des 9èmes Journées Nationales du Réseau Doctoral en Microélectronique, May 2006. ISBN : 978-2-9527172-0-5.

[FFT+06] Olivier Faurax, Laurent Freund, Assia Tria, Traian Muntean, and Frédéric Bancel. A generic method for fault injection in circuits. In *IWSOC 2006 : Proceedings of the 6th IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 211–214. IEEE Circuits and Systems Society, 2006.

[FM07] Olivier Faurax and Traian Muntean. Security analysis and fault injection experiment on AES. In *Actes de la 2ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI 2007)*, June 2007.

[FMR06] Julien Francq, Pascal Manet, and Jean-Baptiste Rigaud. Material emulation of faults on cryptoprocessors. In *Proceedings of Sophia Antipolis forum of MicroElectronics (SAME) 2006*, 2006.

[FTF+07] Olivier Faurax, Assia Tria, Laurent Freund, Frédéric Bancel, and Traian Muntean. PAFI : Outil d'analyse de circuit pour l'accélération de l'injection de fautes en simulation. In *Actes des 10èmes Journées Nationales du Réseau Doctoral en Microélectronique*, May 2007.

[FTFB07] Olivier Faurax, Assia Tria, Laurent Freund, and Frédéric Bancel. Robustness of circuits under delay-induced faults : test of AES with the pafi tool. In *13th IEEE International On-Line Testing Symposium (IOLTS 2007), 8-11 July 2007, Heraklion, Crete, Greece*, pages 185–186. IEEE Computer Society, 2007.

[Gir04] Christophe Giraud. DFA on AES. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[GIY97] Kumar K. Goswami, Ravishankar K. Iyer, and Luke Young. Depend : A simulation-based environment for system level dependability analysis. *IEEE Trans. Comput.*, 46(1) :60–74, 1997.

[GKT89] U. Gunneflo, J. Karlsson, and J. Torin. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In *Proceedings of the 19th International Symposium on Fault Tolerant Computing, (FTCS-19), IEEE, Austin, Texas, USA*, pages 340–347, 1989.

[GS95] Jens Güthoff and Volkmar Sieh. Combining software-implemented and simulation-based fault injection into a single fault injection method. In *FTCS '95 : Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 196–206, Washington, DC, USA, 1995. IEEE Computer Society.

[HZ04] Stefan Hallerstede and Yann Zimmermann. Circuit design by refinement in eventb. In *Forum on Specification and Design Languages - FDL'04, Lille, France*. Pierre Boulet, Sep 2004.

[JAR+94] Eric Jenn, Jean Arlat, Marcus Rimen, Joakim Ohlsson, and Johan Karlsson. Fault injection into VHDL models : The MEFISTO tool. In *Proceedings of the 24th International Symposium on Fault Tolerant Computing, (FTCS-24), IEEE, Austin, Texas, USA*, pages 66–75, 1994.

[KJJ99] P.C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, 1999.

[KKA95] Ghani A. Kanawati, Nasser A. Kanawati, and Jacob A. Abraham. Ferrari : A flexible software-based fault and error injection system. *IEEE Trans. Comput.*, 44(2) :248–260, 1995.

[KLD+94] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo. Using heavy-ion radiation to validate fault-handling mechanisms. *Micro, IEEE*, 14(1) :8–23, 1994.

[Koc96] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. *Advances in Cryptology–CRYPTO*, 96 :104–113, 1996.

[LH92] H. K. Lee and D. S. Ha. Hope : an efficient parallel fault simulator for synchronous sequential circuits. In *DAC '92 : Proceedings of the 29th ACM/IEEE conference on Design automation*, pages 336–340, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[LH00] R. Leveugle and K. Hadjiat. Optimized generation of vhdl mutants for injection of transition errors. In *SBCCI '00 : Proceedings of the 13th symposium on Integrated circuits and systems design*, page 243, Washington, DC, USA, 2000. IEEE Computer Society.

[MHGT92] G. Miremadi, J. Harlsson, U. Gunneflo, and J. Torin. Two software techniques for on-line error detection. *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 328–335, 1992.

[MRCV99] T. Monnier, FM Roche, J. Cosculluela, and R. Velazco. SEU testing of a novel hardened register implemented using standard CMOS technology. *Nuclear Science, IEEE Transactions on*, 46(6) :1440–1444, 1999.

[MRL+06] Y. Monnet, M. Renaudin, R. Leveugle, N. Feyt, P. Moitrel, and F.M.B. Nzenguet. Practical Evaluation of Fault Countermeasures on an Asynchronous DES Crypto Processor. *Proceedings of the 12th IEEE International Symposium on On-Line Testing*, pages 125–130, 2006.

[MRMS94] Henrique Madeira, Mario Zenha Rela, Francisco Moreira, and Joao Gabriel Silva. RIFLE : A general purpose pin-level fault injector. In *European Dependable Computing Conference*, pages 199–216, 1994.

[MS02] S. Morioka and A. Satoh. An Optimized S-Box Circuit Architecture for Low Power AES Design. *Proc. CHES*, pages 172–186, 2002.

[MSS06] Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A generalized method of differential fault attack against AES cryptosystem. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 91–100. Springer, 2006.

[Nic99] M. Nicolaidis. Time Redundancy Based Soft-Error Tolerance to Rescue Nanometer Technologies. *VTS*, 99 :86–94, 1999.

[Nor96] E. Normand. Single event upset at ground level. *Nuclear Science, IEEE Transactions on*, 43(6 Part 1) :2742–2750, 1996.

[OBF+93] J. Olsen, PE Becher, PB Fynbo, P. Raaby, and J. Schultz. Neutron-Induced Single Event Upsets in Static RAMS Observed at 10 KM Flight Altitude. *IEEE TRANSACTIONS ON NUCLEAR SCIENCE*, 40(2), 1993.

[PQ03] Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the AES and KHAZAD. In *Cryptographic Hardware and Embedded Systems − CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[QS01] J.J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA) : Measures and Counter-Measures for Smart Cards. *Proceedings of the International Conference on Research in Smart Cards : Smart Card Programming and Security*, pages 200–210, 2001.

[RAA02] M. Rodriguez, A. Albinet, and J. Arlat. MAFALDA-RT : a tool for dependability assessment of real-time systems. *Dependable Systems and Networks, 2002. Proceedings. International Conference on*, pages 267–272, 2002.

[RM07] B. Robisson and P. Manet. Differential behavioral analysis. In *CHES*, volume 4727, pages 413–426, 2007.

[RSA78] RL Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications*, 1978.

[SA02] S. Skorobogatov and R. Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems − CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[SJPB95] D.T. Smith, B.W. Johnson, J.A. III Profeta, and D.G. Bozzolo. A method to determine equivalent fault classes for permanent and transient faults. In *Proceedings of the IEEE Reliability and Maintainability Symposium, 1995*, pages 418–424, Washington, DC, USA, 1995. IEEE Computer Society.

[Smi85] Gordon L. Smith. Model for delay faults based upon paths. In *Proceedings International Test Conference 1985, Philadelphia, PA, USA, November 1985*, pages 342–351. IEEE Computer Society, 1985.

[Sta99] N.F. Standard. DATA ENCRYPTION STANDARD (DES). *Federal Information Processing Standards Publication*, 1999. http ://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[Sta01] N.F. Standard. Announcing the ADVANCED ENCRYPTION STANDARD (AES). *Federal Information Processing Standards Publication*, 197, 2001. http ://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[STB97] Volkmar Sieh, Oliver Tschäche, and Frank Balbach. Verify : Evaluation of reliability using vhdl-models with embedded fault descriptions. In *FTCS '97 : Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, pages 32–36, Washington, DC, USA, 1997. IEEE Computer Society.

[SU96] E. Soljanin and R. Urbanke. An Efficient Architecture for Implementation of a Multiplier and Inverter in $GF(2^8)$. *Bell-Labs Technical Memo, BLO11217-960308-O8TM*, 1996.

[Ver26] G.S. Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *Journal of the American Institute of Electrical Engineers*, 45 :109–115, 1926.

[VN82] J. Von Neumann. First Draft of a Report on the EDVAC. *The Origins of Digital Computers : Selected Papers*, 1982.

[WOL02] J. Wolkerstorfer, E. Oswald, and M. Lamberger. An ASIC implementation of the AES SBoxes. *Proc. RSA Conference*, 2002.

[WTSI94] Wei Wang, Kishor S. Trivedi, Babubhai V. Shah, and Joseph A. Profeta III. The impact of fault expansion on the interval estimate for fault detection coverage. In *FTCS '94 : Proceedings of the 24th International Symposium on Fault-Tolerant Computing (FTCS '94)*, pages 330–337, Austin, TX, USA, 1994. IEEE Computer Society.

[YJ00] Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Comput.*, 49(9) :967–970, 2000.

[YS96] Charles R. Yount and Daniel P. Siewiorek. A methodology for the rapid injection of transient hardware errors. *IEEE Trans. Comput.*, 45(8) :881–891, 1996.

[Zeb04] Enrique Zebala. Animation sur l'algorithme AES, 2004.

[ZME03] Hamid R. Zarandi, Seyed Ghassem Miremadi, and Alireza Ejlali. Dependability analysis using a fault injection tool based on synthesizability of hdl models. In *DFT '03 : Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 485–492. IEEE Computer Society, 2003.

Annexe C

Publications

```
@InProceedings{of08,
author = {Olivier Faurax and Traian Muntean},
title = {Security analysis of AES using functionality fault model},
booktitle = {Journal of Circuits, Systems, and Computers (À
paraître)},
year = {2008},
}

@InProceedings{ff07,
author = {Olivier Faurax and Julien Francq},
title = {Security of several AES implementations against delay
faults},
booktitle = {Proceedings of the 12th Nordic Workshop on Secure IT
Systems (NordSec 2007)},
year = {2007},
month = {October}
}

@InProceedings{ftfb07,
author = {Olivier Faurax and Assia Tria and Laurent Freund and
Frédéric Bancel},
title = {Robustness of circuits under delay-induced faults : test
of AES with the PAFI tool},
booktitle = {13th IEEE International On-Line Testing Symposium
(IOLTS 2007)},
year = {2007},
pages = {185-186},
}

@InProceedings{of07-2,
```

```
author = {Olivier Faurax and Traian Muntean},
title = {Security analysis and fault injection experiment on AES},
booktitle = {Actes de la 2ème Conférence sur la Sécurité des
Architectures Réseaux et des Systèmes d'Information (SAR-SSI
2007)},
year = {2007},
month = {June}
}

@InProceedings{of07,
author = {Olivier Faurax and Assia Tria and Laurent Freund and
Frédéric Bancel and Traian Muntean},
title = {PAFI : Outil d'analyse de circuit pour l'accélération de
l'injection de fautes en simulation},
booktitle = {Actes des 10èmes Journées Nationales du Réseau
Doctoral en Microélectronique},
year = {2007},
month = {May}
}

@InProceedings{of06-2,
author = {Olivier Faurax and Laurent Freund and Assia Tria and
Traian Muntean and Frédéric Bancel},
title = {A generic method for fault injection in circuits},
booktitle = {IWSOC 2006: Proceedings of the 6th IEEE International
Workshop on System-on-Chip for Real-Time Applications},
year = {2006},
isbn = {1-4244-0898-9},
pages = {211-214},
publisher = {IEEE Circuits and Systems Society},
}


@InProceedings{of06,
author = {Olivier Faurax and Laurent Freund and Frédéric Bancel
and Traian Muntean},
title = {Une méthode générique pour l'injection de fautes dans les
circuits},
booktitle = {Actes des 9èmes Journées Nationales du Réseau
Doctoral en Microélectronique},
isbn = {978-2-9527172-0-5},
note = {ISBN : 978-2-9527172-0-5},
year = {2006},
```

```
month = {May}
}
```