Processeur résistant et résilient aux attaques de fautes et aux attaques par canaux auxiliaires

Gaëtan Leplus

488

N°d'ordre NNT : 2023STET059

THÈSE de DOCTORAT
DE L'UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Membre de l'Université de LYON

École Doctorale N° 488
Sciences Ingénierie et Santé

Spécialité / discipline de doctorat : Microélectronique

Soutenue publiquement/à huis clos le 18/12/2023, par :
Gaëtan Leplus

Processeur résistant et résilient aux attaques en fautes et aux attaques par canaux auxiliaires

---

Devant le jury composé de :

Di Natale, Giorgio	DR	Université Grenoble Alpes – CNRS - TIMA	Rapporteur
Pillement, Sébastien	Pr	Université de Nantes - IETR	Rapporteur
Gogniat, Guy	Pr	Université de Nantes – STICC	Examinateur
Heydemann, Karine	IR-HDR	THALES	Examinatrice
Bossuet, Lilian	Pr	Université Jean Monnet St Etienne – LabHC	Directeur de thèse
Savry, Olivier	IR	CEA	Encadrant de thèse

RÉSUMÉ

Dans le paysage technologique actuel, l'internet des objets (IoT) a émergé comme un élément omniprésent, engendrant néanmoins des défis majeurs en matière de sécurité. Les attaques par injection de fautes et par canaux auxiliaires sont particulièrement préoccupantes, visant les faiblesses systémiques et compromettant l'intégrité et la confidentialité des données. Les processeurs, étant les pierres angulaires des systèmes informatiques, sont cruciaux pour la sécurisation de l'IoT.

Cette thèse se focalise sur la sécurisation du pipeline des processeurs pour contrer ces menaces. L'importance de cette recherche est mise en évidence par la nécessité d'élaborer des mécanismes de sécurité robustes au niveau du processeur, le noyau de tout calcul et contrôle. Plusieurs contre-mesures sont proposées pour renforcer la résilience des différentes parties du processeur face aux attaques.

Pour sécuriser le chemin de données, une méthode de tag d'intégrité est proposée. Compatible avec les techniques de masquage traditionnelles, cette méthode vise à garantir l'intégrité des données tout au long du pipeline du processus, et ce, avec un surcoût réduit. Cette approche permet de protéger le système contre les altérations de données.

Pour le chemin d'instructions, un mécanisme de masquage de l'instruction en cours est proposé, où un masque est généré en fonction de l'instruction précédente. Cette technique innovante permet une sécurisation efficace des instructions avec un surcoût très faible, se posant ainsi comme une solution viable pour prévenir les fuites d'informations et garantir la confidentialité du code exécuté.

Quant au procédé de désynchronisation, il introduit une méthode robuste permettant d'insérer des instructions factices de manière plus efficace que les techniques actuelles. En introduisant un degré d'aléatoire et de désynchronisation dans l'exécution des instructions, ce procédé entrave les tentatives des attaquants cherchant à analyser ou à manipuler le flux d'exécution.

Ces contre-mesures, en ciblant les composantes clés du processeur, contribuent à une amé-

lioration notable de la sécurité des systèmes IoT. Elles s'attaquent aux racines des vulnérabilités, offrant ainsi un niveau de protection renforcé contre une gamme variée d'attaques.

De plus, cette recherche souligne l'importance de repenser la conception des processeurs en intégrant la sécurité dès les premières étapes, afin de relever les défis posés par les attaques les plus récentes.

Mot clés : canaux auxiliaires, injections de fautes, contremesures, processeur, RISCV

ABSTRACT

In today's technological landscape, the Internet of Things (IoT) has emerged as a ubiquitous element, yet it brings major security challenges. Fault injection and side-channel attacks are of particular concern, targeting systemic weaknesses and compromising data integrity and confidentiality. Processors, as the cornerstones of computing systems, are crucial in securing the IoT.

This thesis focuses on securing the processor pipeline to counter these threats. The significance of this research is highlighted by the need to develop robust security mechanisms at the processor level, the core of all computation and control. Several countermeasures are proposed to enhance the resilience of different parts of the processor against attacks.

To secure the data path, an integrity tagging method is proposed. Compatible with traditional masking techniques, this method aims to ensure data integrity throughout the processing pipeline, with minimal overhead. This approach helps protect the system against data tampering.

For the instruction path, a mechanism for masking the current instruction is proposed, where a mask is generated based on the previous instruction. This innovative technique enables effective instruction security with very low overhead, thus presenting itself as a viable solution for preventing information leakage and ensuring the confidentiality of executed code.

Regarding the desynchronization process, it introduces a robust method for inserting dummy instructions more efficiently than current techniques. By introducing a degree of randomness and desynchronization in the execution of instructions, this process hinders attackers' attempts to analyze or manipulate the execution flow.

These countermeasures, by targeting key components of the processor, contribute to a notable improvement in the security of IoT systems. They address the roots of vulnerabilities, thus providing enhanced protection against a wide range of attacks.

Furthermore, this research underscores the importance of rethinking processor design by incorporating security from the early stages, in order to meet the challenges posed by the most recent attacks.

Keywords: side channels, fault injections, countermeasures, processor, RISCV

TABLE OF CONTENTS

TABLE DES FIGURES

LISTE DES TABLEAUX

1

Introduction

L'essor exponentiel des objets connectés s'étend à une multitude de domaines, y compris la domotique, l'industrie manufacturière, et le secteur de la santé. Ces dispositifs offrent des avantages substantiels, tels que la gestion optimisée des ressources énergétiques, l'automatisation des tâches répétitives, et une amélioration significative de la qualité de vie grâce à des applications de suivi de la santé, par exemple. Toutefois, cette expansion rapide soulève des questions cruciales en matière de cybersécurité, augmentant la vulnérabilité de nos sociétés à une gamme étendue de menaces.

L'intégration croissante de ces objets connectés dans notre quotidien a pour effet d'élargir considérablement la surface d'attaque accessible aux cybercriminels. Chaque dispositif nouvellement connecté représente un point d'entrée potentiel pour des attaques malveillantes. En raison de contraintes budgétaires, de limitations de taille et de considérations relatives à la consommation d'énergie, ces objets sont souvent moins bien sécurisés que les systèmes informatiques traditionnels. Par exemple, ils peuvent être dépourvus de pare-feu efficaces ou de mécanismes d'authentification robustes, rendant ainsi le réseau entier vulnérable.

L'absence de normes de sécurité uniformes et réglementées pour les objets connectés exacerbe le problème. Les fabricants, souvent plus préoccupés par la rapidité de mise sur le marché que par la sécurité, peuvent omettre des mesures de sécurité fondamentales. Cela inclut le chiffrement robuste des données transmises et stockées, ainsi que la mise à jour régulière du firmware pour corriger les vulnérabilités connues. Cette négligence rend ces dispositifs particulièrement exposés à des attaques telles que l'interception de données ou l'installation de logiciels malveillants.

Les risques associés à la cybersécurité des objets connectés ne se cantonnent pas à des violations de la vie privée ou à des fuites de données financières. Dans des domaines sensibles comme la santé ou les infrastructures critiques (énergie, transport, etc.), une attaque réussie peut avoir des conséquences catastrophiques. Par exemple, le piratage d'un dispositif médical implantable, comme un stimulateur cardiaque, pourrait mettre directement en danger la vie du patient. De même, une attaque sur les systèmes de contrôle d'une centrale électrique pourrait entraîner des pannes massives, affectant ainsi la sécurité et le bien-être de millions de personnes.

1.1 Attaques physiques

Quand on évoque la cybersécurité, l'attention se porte souvent sur les aspects liés à la sécurité des réseaux et aux menaces logicielles, telles que les virus, les ransomwares et autres malwares. Cependant, il est crucial de ne pas négliger la couche matérielle qui constitue la fondation de tout système informatique, y compris les objets connectés. Cette couche matérielle peut être la cible d'attaques spécifiques ou servir de tremplin pour des attaques logicielles plus sophistiquées.

Les attaques sur la couche matérielle peuvent prendre plusieurs formes. L'une des plus connues est l'attaque par canal auxiliaire, qui exploite les informations involontairement divulguées par le matériel, comme la consommation d'énergie ou les émissions électromagnétiques. Par exemple, une attaque par analyse de la consommation électrique peut permettre de récupérer des clés de chiffrement sur un dispositif. De même, les attaques par injection de fautes, qui induisent des erreurs dans le matériel pour en déduire des informations sensibles ou pour provoquer un comportement imprévu, sont également possibles.

Les défauts matériels peuvent également être exploités pour faciliter ou amplifier des attaques logicielles. Par exemple, les vulnérabilités matérielles comme Spectre et Meltdown ont montré comment des défauts dans la conception des processeurs peuvent être exploités pour accéder à des données sensibles à travers des attaques logicielles. Ces vulnérabilités matérielles peuvent rendre les mécanismes de sécurité logicielle inefficaces, car elles opèrent à un niveau plus bas que le logiciel.

La complexité des systèmes modernes, où le matériel et le logiciel sont étroitement intégrés, rend la tâche encore plus difficile. Dans un objet connecté, le firmware, le système

d'exploitation et les applications coexistent avec divers composants matériels, tels que les processeurs, les capteurs et les interfaces réseau. Cette interdépendance signifie que la compromission de la couche matérielle peut avoir des répercussions en cascade sur l'ensemble du système.

1.2 Le cœur matériel des objets connectés : les processeurs

Les processeurs constituent la brique fondamentale du matériel dans l'écosystème des objets connectés. Ils sont le cœur de toute opération, gérant l'exécution des instructions, le traitement des données et la communication avec d'autres composants matériels et logiciels. Dans un monde où l'Internet des Objets (IoT) est en pleine expansion, la performance, la sécurité et l'efficacité des processeurs deviennent des critères de plus en plus cruciaux. La qualité du processeur détermine en grande partie la fiabilité, la sécurité et l'efficacité de l'ensemble du système. Par conséquent, le choix d'une architecture de processeur appropriée est une décision stratégique qui a des implications profondes sur la fonctionnalité et la sécurité des objets connectés.

Ainsi, en plus des exigences liées à la consommation énergétique et au coût de ces dispositifs, de nouvelles contraintes de sécurité émergent avec ces usages. Ces composants peuvent être amenés à stocker, exécuter et collecter des données sensibles, et les situations où des acteurs malveillants ont un accès physique aux dispositifs deviennent de plus en plus probables. Lorsqu'il s'agit de sécurité, les principaux objectifs à respecter sont les suivants :

- Confidentialité : Le processeur doit préserver la confidentialité des instructions et des données utilisées.
- Intégrité : Les instructions, les données manipulées et l'état du processeur ne doivent pas subir de modifications non autorisées entre le stockage et leur traitement. Ces modifications peuvent être accidentelles, illicites ou malveillantes. En d'autres termes, les éléments considérés doivent être exacts et non altérés.
- Authenticité : Le processeur doit être capable de vérifier l'origine des données. Le niveau de granularité de l'origine peut varier en fonction des cas d'utilisation. L'authenticité est souvent assurée par une signature sur les données, qui permet également de garantir leur intégrité.

– Disponibilité : Le processeur doit assurer son fonctionnement en tout temps.

Au-delà de ces notions de sécurité, il est essentiel de les articuler de manière adéquate, en particulier en envisageant les conséquences de pertes d'intégrité, de confidentialité, d'authenticité et de disponibilité. Par exemple, on peut exiger que le processeur soit capable de se rétablir après ces perturbations et retrouver un fonctionnement nominal tout en ayant réparé ses erreurs. On parle alors de résilience. Celle-ci passe par trois stades : la protection, la détection et le rétablissement.

Les travaux menés au cours de cette thèse visent à garantir ces quatre propriétés face aux attaques par canaux auxiliaires et par injection de fautes dans le contexte des processeurs aussi appelé unité centrale de traitement (CPU).

Parmi les architecture de CPU utilisable dans l'IoT, depuis quelques années, un nouveau standard ouvert a fait son apparition dans le paysage de la conception de processeurs : le RISC-V. Ce standard est né en réponse aux défis croissants liés aux coûts et à la complexité des architectures de processeurs traditionnelles. Il offre une alternative flexible et économique, particulièrement attrayante pour les objets connectés et autres dispositifs où les contraintes de coût, de taille et de consommation d'énergie sont prédominantes.

1.3 Description du RISC-V

L'un des principaux avantages du RISC-V réside dans sa nature ouverte. Contrairement aux architectures propriétaires, le RISC-V permet aux concepteurs de personnaliser le processeur pour des besoins spécifiques, ce qui peut inclure l'intégration de fonctionnalités de sécurité avancées. Cette flexibilité peut être particulièrement bénéfique pour aborder les problématiques de cybersécurité au niveau matériel, en permettant par exemple l'implémentation de mécanismes de chiffrement ou de vérification d'intégrité directement dans le silicium.

L'adoption croissante du RISC-V a des implications significatives pour la cybersécurité. D'une part, la possibilité de personnaliser l'architecture permet d'intégrer des mesures de sécurité spécifiques dès la phase de conception. D'autre part, étant un standard ouvert, il est soumis à un examen minutieux de la communauté, ce qui peut accélérer la découverte et la correction de vulnérabilités potentielles.

Cependant, cette ouverture pose aussi des défis. Le code et les spécifications étant acces-

sibles à tous, les attaquants ont également la possibilité d'étudier en détail l'architecture pour y trouver des failles. De plus, la diversité des implémentations, rendue possible par la flexibilité du standard, peut entraîner une hétérogénéité en matière de sécurité, rendant difficile l'établissement de normes de sécurité uniformes.

L'écosystème RISC-V est un environnement riche et diversifié qui englobe une multitude d'acteurs et de technologies. Il s'étend bien au-delà des simples fabricants de puces pour inclure des fournisseurs de logiciels, des institutions académiques, des centres de recherche, et même des passionnés de l'informatique.

1.3.1 Acteurs clés

- **Fabricants de puces** : des entreprises comme SiFive, Western Digital et d'autres contribuent à la conception et à la fabrication de puces basées sur l'architecture RISC-V.
- **Fournisseurs de logiciels** : des entreprises de logiciels et des communautés open-source développent des compilateurs, des débogueurs et d'autres outils essentiels pour le développement sur RISC-V.
- **Institutions académiques et centres de recherche** : de nombreuses universités et centres de recherche contribuent à l'évolution de l'architecture RISC-V, souvent en partenariat avec l'industrie. L'OpenHW Group,par exemple, rassemble des membres issus de l'industrie, de la recherche académique et du secteur public
- **RISC-V International Association** : Cette organisation joue un rôle central en promouvant l'adoption du standard RISC-V, en standardisant les nouvelles extensions et en fournissant des ressources éducatives.

L'écosystème est également riche en outils de développement, allant des compilateurs et débogueurs aux simulateurs et aux plates-formes de vérification. Des systèmes d'exploitation spécialement conçus ou adaptés pour RISC-V sont également disponibles, y compris des versions de Linux et de systèmes d'exploitation temps réel (RTOS).

1.3.2 Profils de base

- **RV32I** : Il s'agit de l'ensemble d'instructions de base sur des entiers de 32 bits. C'est le plus simple et sert souvent de fondement aux extensions plus complexes.

- **RV64I** : Cet ensemble d'instructions de 64 bits est destiné aux systèmes nécessitant une plus grande capacité de mémoire et de calcul. Il est souvent utilisé dans les serveurs et les applications de traitement de données de grande envergure et les processeurs d'applications pour l'embarqué comme les smartphones.
- **RV128I** : Bien que moins courant, ce profil de 128 bits est conçu pour des applications nécessitant des opérations sur de très grands ensembles de données, comme certaines tâches de calcul scientifique.

Chaque profil peut être étendu avec des extensions standardisées pour ajouter des fonctionnalités. Par exemple, l'extension "M" ajoute des instructions de multiplication et de division, tandis que l'extension "F" ajoute le support du calcul en virgule flottante. Ces extensions permettent une personnalisation fine des processeurs RISC-V pour des applications spécifiques, offrant ainsi une flexibilité sans précédent aux concepteurs de systèmes.

Parmi les nombreuses implémentations de RISC-V, le CV32E40P se distingue pour plusieurs raisons qui en font une option particulièrement attrayante pour diverses applications et pour les travaux de cette thèse.

1.4 Introduction au CV32E40P

Le CV32E40P est un processeur basé sur l'architecture RISC-V, spécifiquement conçu pour les systèmes embarqués et les applications nécessitant un équilibre entre haute performance et faible consommation d'énergie. Développé par l'OpenHW Group, ce processeur est une implémentation du profil RV32E de l'architecture RISC-V, qui est une version optimisée pour les systèmes embarqués. Le "E" dans RV32E signifie "Embedded", indiquant que ce profil est particulièrement adapté aux dispositifs avec des contraintes de ressources.

Le CV32E40P est doté d'une architecture pipeline en 4 étapes, voir figure 1.1, comprenant les étapes Fetch, Decode, Execute et Write-back qui seront présentées dans la suite . De plus, le CV32E40P prend en charge diverses extensions RISC-V, offrant ainsi une grande flexibilité pour des opérations arithmétiques, des opérations en virgule flottante et d'autres fonctionnalités avancées.

Ce processeur trouve des applications dans une variété de domaines, allant de la domotique et de la santé connectée à l'industrie et aux réseaux de capteurs. Sa conception modulaire et sa faible consommation d'énergie en font un choix idéal pour les dispositifs

alimentés par batterie, tandis que sa haute performance le rend apte pour des applications nécessitant des calculs intensifs.

1.4.1 Étape Fetch

Cette étape est la première du pipeline et est responsable de la récupération des instructions à partir de la mémoire. Elle prépare également les instructions pour le décodage et l'exécution ultérieurs. Elle est composée des trois blocs suivants :

- **Prefetch Buffer** : Stocke les instructions récupérées en avance pour minimiser les temps d'attente dus aux accès à la mémoire.
- **Aligner** : Ajuste les instructions pour qu'elles soient correctement alignées avant le décodage.
- **Compress Decoder** : Décode les instructions compressées en leur forme standard pour un traitement ultérieur.

1.4.2 Étape Decode

Cette étape prend les instructions récupérées et les décode en opérations que le processeur peut exécuter. Elle est également responsable de la gestion des boucles matérielles et de la lecture des registres. Elle est composée des trois blocs suivants :

- **HWLoop** : Détecte et gère les boucles matérielles pour améliorer l'efficacité du pipeline.
- **Decoder** : Traduit les instructions en signaux de contrôle pour les étapes suivantes du pipeline.
- **Register File** : Lit les valeurs des registres nécessaires pour l'exécution de l'instruction.

1.4.3 Étape Execute

Cette étape est le cœur du pipeline où les opérations arithmétiques, logiques et de mémoire sont effectuées en fonction des instructions décodées. Elle est composée des quatre blocs suivants :

- **CSR (Control Status Registers)** : Gère les registres de contrôle et d'état.

- **ALU (Arithmetic Logic Unit)** : Effectue les opérations arithmétiques et logiques.
- **Mult (Multiplier)** : Effectue les opérations de multiplication.
- **LSU (Load/Store Unit)** : Gère les opérations de chargement et de stockage en mémoire.

1.4.4 Étape Write-back

Dans cette étape, les résultats des opérations sont écrits dans les registres de destination, achevant ainsi le cycle de vie de l'instruction.

1.4.5 Contrôleur

Le contrôleur orchestre le fonctionnement des différentes étapes du pipeline, en gérant les dépendances entre les instructions et en s'assurant que le pipeline fonctionne de manière fluide.

1.4.6 Sleep Unit

La Sleep Unit permet au processeur d'entrer dans des états de faible consommation d'énergie lorsqu'il n'est pas sollicité, contribuant ainsi à l'efficacité énergétique globale du système.

En somme, le CV32E40P est une implémentation robuste et flexible de l'architecture RISC-V, optimisée pour les systèmes embarqués et dotée de nombreuses fonctionnalités pour améliorer à la fois la performance et l'efficacité énergétique. C'est pour ces raisons que nous l'avons choisi comme cible d'implantation de nos travaux.

1.5 Attaques par Canaux Auxiliaires et par Injections de Faute sur le Pipeline de Processeurs

Les processeurs modernes, y compris ceux basés sur l'architecture RISC-V comme le CV32E40P, sont confrontés à diverses menaces de sécurité. Parmi ces menaces, les attaques par canaux auxiliaires et les attaques par injections de faute sont particulièrement préoccupantes en raison de leur capacité à exploiter les caractéristiques physiques et les comportements du pipeline du processeur.

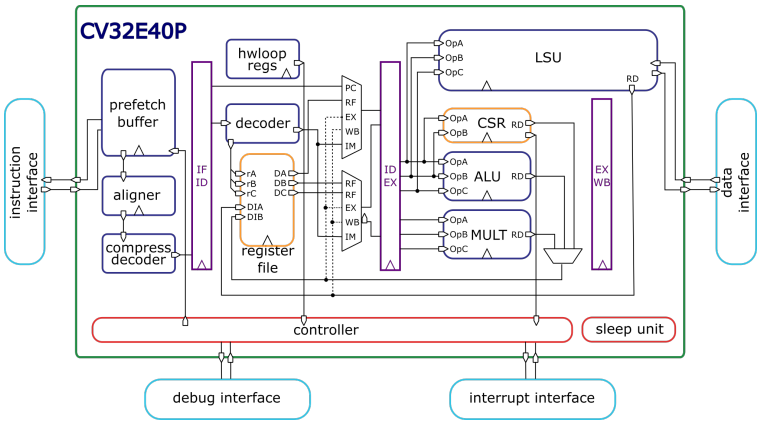

Figure 1.1 : Microarchitecture du CPU CV32E40P.

1.5.1 Attaques par Canaux Auxiliaires

Les attaques par canaux auxiliaires exploitent les informations involontairement divulguées par le processeur pendant son fonctionnement normal. Ces informations peuvent inclure des mesures de la consommation d'énergie, des émissions électromagnétiques ou même le temps nécessaire pour exécuter certaines instructions. Par exemple, une attaque par canal auxiliaire peut analyser la consommation d'énergie du processeur pendant l'exécution d'un algorithme de chiffrement pour en déduire la clé secrète.

Dans le contexte du pipeline, ces attaques peuvent être encore plus pernicieuses. Les différentes étapes du pipeline, comme Fetch, Decode et Execute, peuvent avoir des profils de consommation d'énergie distincts. Un attaquant peut utiliser ces différences pour déterminer quel type d'instruction est en cours d'exécution, offrant ainsi des indices sur les données traitées.

1.5.2 Attaques par Injections de Faute

Les attaques par injections de faute, en revanche, sont plus actives. Elles impliquent l'introduction délibérée d'erreurs dans le processeur pour en perturber le fonctionnement normal. Ces erreurs peuvent être introduites par divers moyens, tels que des impulsions laser, des variations de la tension d'alimentation ou des interférences électromagnétiques. Une fois que la faute est injectée, elle peut provoquer des comportements imprévus, comme des sauts conditionnels incorrects ou des erreurs de calcul.

Dans un pipeline, les effets d'une injection de faute peuvent être amplifiés. Par exemple, une faute injectée dans l'étape Fetch pourrait affecter les instructions qui sont actuelle-

ment dans les étapes Decode et Execute, entraînant une cascade d'erreurs qui peuvent être difficiles à détecter et à corriger. De plus, les mécanismes de sécurité intégrés, comme les registres de contrôle d'état (CSR), pourraient également être compromis, rendant le système encore plus vulnérable.

1.5.3 Contre-mesures

Ces deux types d'attaques peuvent être menées conjointement, en utilisant des canaux auxiliaires pour identifier des points d'intérêt pour l'injection de fautes. Toutefois, leurs contre-mesures classiques sont souvent contradictoires. Les canaux auxiliaires nécessitent le masquage des données, qui consiste à éviter de manipuler directement une valeur sensible, mais plutôt à utiliser un ensemble de variables (appelées « shares »). Un attaquant doit obtenir toutes les valeurs des shares pour extraire des informations significatives. Cette répartition en shares offre davantage de possibilités d'attaques d'un point de vue de l'injection de fautes, car n'importe quel share peut être ciblé.

Les contre-mesures contre l'injection de faute reposent généralement sur le principe de redondance, qu'elle soit temporelle, physique ou informationnelle. Cependant, cette redondance peut entraîner de nouvelles fuites exploitables par des canaux auxiliaires.

Des méthodes mixtes ont été proposées dans l'état de l'art, notamment M&M [Mey+19], qui utilise les propriétés spécifiques des opérations de l'AES pour créer une redondance capable de vérifier les calculs arithmétiques tout en étant compatible avec le masquage. M&M propose une approche qui combine les avantages des contre-mesures classiques contre les canaux auxiliaires et les injections de fautes, tout en minimisant leurs inconvénients respectifs. Cette méthode permet d'assurer une sécurité renforcée contre les deux types d'attaques, en exploitant les caractéristiques intrinsèques de l'AES pour résister aux attaques par canaux auxiliaires et injections de fautes simultanément.

1.6 Sécurité recherchée et organisation de la thèse

Les problématiques abordées dans cette thèse visent à identifier et à contrer les différentes attaques qui peuvent cibler le pipeline d'un processeur généraliste en exploitant les spécificités de chaque partie attaquée. La question centrale est de déterminer s'il est possible d'obtenir un résultat similaire à M&M pour un processeur généraliste, et d'identifier les techniques de masquage les plus adaptées à ce type de composant.

La protection contre les fautes et les fuites, y compris les injections de fautes multiples et les canaux auxiliaires sur les données manipulées par le CPU, est une préoccupation majeure. Cette thèse vise à adresser l'ensemble du pipeline en tenant compte des contraintes spécifiques à chaque partie.

En matière de contremesures, l'objectif est de proposer des solutions légères et rapides, qui ne se limitent pas à l'intégrité des données, mais englobent également leur authenticité. L'importance de ces contremesures réside dans leur capacité à prévenir les attaques tout en maintenant une latence faible. Des méthodes de masquage low-latency sont également étudiées pour rendre l'utilisation de masque réaliste dans le cadre d'un pipeline de processeur.

Il est effectivement crucial de souligner que les modèles d'attaquants varient considérablement en fonction de la partie du pipeline de traitement ciblée. Les attaques sur les instructions et celles sur les données sont intrinsèquement différentes et nécessitent des contre-mesures spécifiques. Cette thèse se concentre principalement sur la protection des données et des circuits de traitement, plutôt que sur la confidentialité des instructions ou le masquage du comportement du processeur.

Des modèles d'attaquants spécifiques sont élaborés pour chaque composante du pipeline, permettant ainsi une compréhension plus approfondie des menaces potentielles et des contre-mesures nécessaires. Ces modèles servent de base pour le développement de solutions de sécurité adaptées, qui sont ensuite évaluées pour leur résilience face aux attaques par canaux auxiliaires et par injections de fautes.

Le chapitre 2 dresse un panorama des principales attaques exploitant les canaux auxiliaires et les injections de fautes, ainsi que leurs contre-mesures respectives. Il offre une analyse de l'état actuel de la recherche dans ces domaines clés pour la sécurité informatique.

Le chapitre 3 s'attache plus spécifiquement au chemin de données du processeur, en mettant en exergue les contre-mesures pouvant être appliquées spécifiquement à cette composante essentielle de l'architecture des processeurs. L'objectif est de présenter des solutions adaptées à la protection de cette partie de la machine contre les menaces potentielles.

Le chapitre 4, quant à lui, se focalise sur le chemin d'instruction et le contrôle, en proposant une revue de l'état de l'art en matière d'attaques et de contre-mesures pour ces composants. Le chapitre propose également des contremesures pour sécuriser le chemin d'instruction et la logique de contrôle.

Dans le chapitre 5, l'attention est portée sur le processeur dans son ensemble. Une contremesure globale est présentée, impactant le fonctionnement général du processeur et prévenant une grande variété d'attaques. En outre, le chapitre expose la manière dont toutes les mesures de sécurité développées dans les chapitres précédents peuvent être intégrées dans un seul et même circuit, à savoir un processeur RISC-V 32 bits CV32E40P sur ASIC.

Enfin, le chapitre 6 apporte la conclusion générale de ce manuscrit, synthétisant les travaux entrepris pour renforcer la sécurité d'une architecture de processeur. Ce chapitre final récapitule les principales découvertes, met en perspective les contributions de la thèse et suggère des pistes pour de futures recherches.

1.7 Contribution

Les travaux présentés dans le chapitre 3 ont donné lieu à deux brevets distincts. Le premier brevet porte sur le concept de permutation dépendant d'une clé, tandis que le second est axé sur la permutation choisie pour faciliter les opérations arithmétiques. Ces travaux font aussi l'objet du soumission le 15 octobre 2023 à la conférence CHES sous le nom de "AKHACIA : Arborescent Keyed Homomorphic tags for Confidentiality, Integrity and Authenticity of data in CPU pipeline"

Le chapitre 4, qui traite de la sécurisation des instructions par la génération de masque à partir de l'instruction précédente, a été publié lors de la conférence DSD2022 [LSB22b]. De plus, cette recherche est également présentée dans un brevet déposé.

La contremesure présenté en chapitre 5 portant sur l'insertion de cycles factices pour créer une désynchronisation temporelle, a été brevetée et a fait l'objet d'une publication lors de la conférence HOST2022 [LSB22a].

2

État de l'art

Dans ce chapitre, nous abordons les types d'attaques auxquelles notre travail cherche à répondre. Nous débutons en examinant les attaques par canaux auxiliaires, leurs vecteurs de fuite et les principales méthodes d'exploitation. Nous examinons ensuite rapidement divers types de contre-mesures avant d'évoquer l'évaluation des circuits. Nous discuterons par la suite des attaques par injections de fautes. Nous définirons d'abord les différentes façons d'injecter des erreurs, et établirons ensuite notre modèle de fautes utilisé tout au long de la thèse. Nous étudierons également diverses méthodes de tests et finirons par décrire différents types de contre-mesures contre les attaques par injection de fautes.

2.1 Attaque canaux auxiliaires

Une attaque par canal auxiliaire cherche à exploiter les failles de l'implémentation matérielle. Ce type d'attaque ne remet pas en cause l'intégrité théorique de l'implémentation, mais vise à récupérer de l'information par des moyens détournés. En effet, lors du traitement des données, l'alternance entre différents états nécessite du temps et une dissipation minimale d'énergie dont les variations peuvent être analysées par l'attaquant.

2.1.1 Méthode d'analyse par canaux auxiliaires

Les canaux auxiliaires sont nombreux, étant donné que tout calcul est le résultat de composants électroniques produisant de la chaleur, une consommation électrique, une émanation électromagnétique et un temps d'exécution propre. Ces fuites, difficiles à contrôler pour le concepteur, peuvent dépendre des données manipulées et permettre à un attaquant d'extraire de l'information.

Fuite par consommation

L'une des sources principales de fuite d'information par canal auxiliaire réside dans la consommation énergétique des circuits électroniques. Cette consommation se décompose en deux parties : une composante dynamique et une composante statique.

La consommation dynamique, ou Dynamic Power Dissipation (DPD), est la principale source de fuite par canal auxiliaire. Elle se produit lors d'un changement d'état logique et est composée de deux éléments : le chargement et le déchargement de la capacité de charge, perceptible lors des transitions de 0-1 ou de 1-0, ainsi que le court-circuit dû à un signal d'entrée non nul [Kor16].

La consommation statique, ou Static Power Dissipation (SPD), est composée de six mécanismes illustrés à la Figure 2.1 :

- I_1 Consommation dans la jonction p-n,
- I_2 Fuite par subthreshold,
- I_3 Tunneling à travers la porte oxyde,
- I_4 Injection de Hot carriers dans le substrat de la porte oxyde,
- I_5 Fuite du drain induite par la gate,

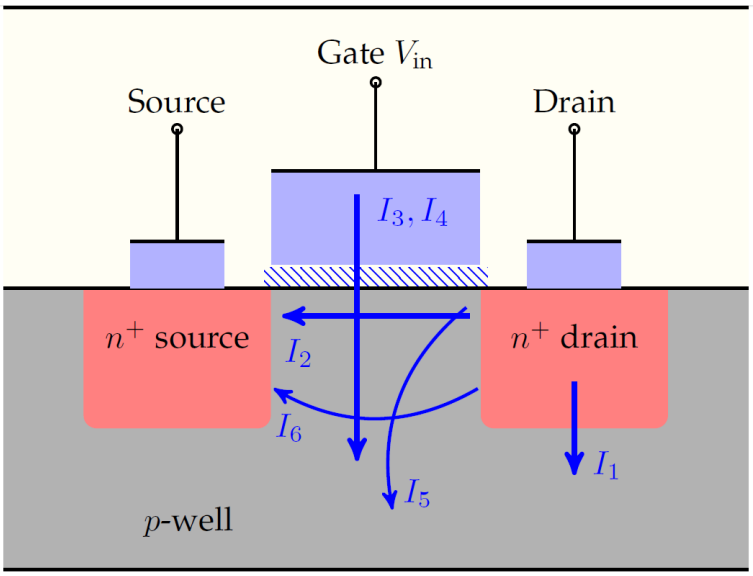

Figure 2.1 : Résumé des mécanismes de SPD.

- I_6 Courant de punch-through.

Les fuites I_2, I_5, I_4, I_6 n'ont lieu que lorsque le circuit est dans l'état "off", ce qui signifie que la SPD permet l'extraction de données à partir de ces fuites, alors que I_1 et I_3 sont indépendants des données manipulées.

Fuite électromagnétique

Les émissions électromagnétiques proviennent du courant circulant lors du traitement des données, du contrôle ou de toute autre partie du circuit. Ces courants peuvent être volontaires ou involontaires. Un courant peut également influencer les émanations d'autres composants par des phénomènes de couplage ou en fonction de la géométrie du circuit. La principale différence avec les fuites de consommation est que les fuites électromagnétiques sont locales, tandis que les fuites de consommation représentent l'agrégation totale de toutes les consommations. De plus, elles permettent de différencier le passage des bits de 0 à 1 et de 1 à 0. Deux types d'émanations sont à distinguer, les directes et les indirectes :

– Les émanations directes, qui dépendent directement de la circulation du courant. Il s'avère souvent que les composants avec les fréquences les plus élevées sont les plus utiles à l'attaquant, car le bruit et les interférences se situent principalement en basse fréquence. Dans les circuits complexes, il peut être difficile d'isoler les émanations directes, nécessitant des sondes très précises et très proches de la source du signal. Il

est même préférable de décapsuler le circuit pour obtenir des résultats plus précis.

– Avec la miniaturisation et la complexification des circuits, les phénomènes de couplage électromagnétique et électrique sont de plus en plus significatifs en raison de la proximité des composants. Ces émanations se manifestent sous forme de modulation de signaux porteurs. Le signal d'horloge est un signal porteur fortement modulé par le circuit et constitue donc un vecteur d'attaque important. Cette modulation peut être une modulation d'amplitude s'il y a un couplage non linéaire entre le signal porteur et le signal de données. La donnée peut être extraite grâce à une démodulation d'amplitude. Le couplage de circuit peut également entraîner une modulation de phase. Dans ce cas, une démodulation de phase est nécessaire. L'étude des émanations indirectes peut permettre d'obtenir de meilleurs résultats que les signaux directs, car elles se propagent mieux dans le circuit.

Autres fuites

Le temps d'exécution est un canal de fuite très important, que ce soit en termes de temps d'accès des caches [Ber05], permettant des attaques de type Spectre et Meltdown, ou en termes de temps d'exécution, qui a permis de compromettre certaines implémentations de RSA [Koc96].

La température du circuit peut, de la même manière que la consommation, fournir des informations sur l'état et les données en cours de traitement. Comme pour les fuites électromagnétiques, la température permet d'extraire des informations locales. Cependant, l'augmentation de la température n'est pas instantanée et il faut plusieurs millisecondes, voire secondes, pour observer des fuites significatives. Cela limite les attaques aux systèmes qui répètent en boucle une opération critique [HS13a]. Pour des raisons de précision, il est nécessaire de décapsuler le circuit pour le sonder. Les fuites acoustiques sont peu exploitées, cependant dans [GST17] il a été démontré que ces fuites, dans le contexte des processeurs Intel pour ordinateurs portables, permettaient de déterminer les instructions en cours d'exécution. Selon cet article, les fuites sont bien des vibrations des composants électriques du processeur, bien qu'il soit difficile de les caractériser précisément. Cette attaque a permis d'extraire les clés d'un chiffrement RSA.

2.1.2 Modèle de fuite

Les fuites d'informations qui surviennent dans un circuit peuvent être évaluées et modélisées pour une meilleure compréhension et exploitation. Ces fuites peuvent être caractérisées en deux groupes principaux.

Le premier groupe de fuites dépend de l'état actuel du matériel, c'est-à-dire le nombre d'états à 1. Ce type de fuite est souvent modélisé par le poids de Hamming, qui compte le nombre de bits à 1 dans une représentation binaire.

Le second groupe de fuites est basé sur la transition entre deux états pour en quantifier la différence. Plus précisément, le nombre de transitions entre ces deux états est calculé. Ce type de fuite est modélisé par la distance de Hamming, qui mesure le nombre de bits différents entre deux représentations binaires.

Cependant, il est important de prendre en compte de nombreux facteurs qui peuvent bruiter la mesure, notamment les activités qui s'effectuent en parallèle de la donnée cible, le bruit de l'environnement, mais aussi celui de la mesure, et enfin les paramètres extérieurs tels que la température, les jitters d'horloge ou d'alimentation, etc.

Même s'il est impossible d'éliminer physiquement ces bruits, la loi des grands nombres offre une certaine réassurance. Cette loi stipule qu'à mesure que le nombre de variables distribuées de manière identique et aléatoire augmente, leur moyenne d'échantillon se rapproche de leur moyenne théorique. Ce phénomène permet de distinguer deux valeurs, même si le bruit est significatif, à condition de disposer d'un nombre suffisant de données.

La table 2.1, tirée de [Kor16], donne un aperçu des informations qui peuvent être obtenues à partir des fuites des canaux auxiliaires. En plus des données binaires, telles que le poids de Hamming qui est le nombre de bits qui sont différents de zéro (Hamming weight). La distance de Hamming entre deux chaînes de bits de même longueur (Hamming distance) qui est définie comme le nombre de positions où les symboles correspondants sont différents. la fuite du canal auxiliaire peut fournir des informations sur le moment et l'emplacement spatial d'une donnée ciblée. Cela signifie que les canaux auxiliaires peuvent être utilisés pour la rétro-ingénierie [Tan+12]. les differents canaux auxiliaire comparé sont les fuites par temps d'execution (Time), par fuite en puissance statique(SPD - Static Power Dissipation), en puissance dynamique (Dynamic Power Dissipation), par fuite électromagnétique (EM - Electromagnetic Analysis), par analyse de densité spectrale de puissance (PSD - Power Spectrum Density) et par température (TSA - Temperature Side-Channel)

	Time	SPD	DPD	EM	PSD	TSA
Hamming weight	√	√	√	√	√	√
Hamming distance	√	×	√	√	√	×
Temporal Location	×	×	√	√	×	×
Spatial location	×	×	×	√	×	√

Table 2.1 : Tableau récapitulatif des fuites extrait de [Kor16].

2.1.3 Exploitation des fuites

Attaques simples

Les attaques par analyse de consommation d'énergie les plus simples, ou Simple Power Attack (SPA), exploitent des schémas ou des dispositions qui dépendent de la clé, à partir d'une seule trace de fuite. Ces traces pourraient correspondre à l'enregistrement de la consommation électrique d'un dispositif pendant un chiffrement ou une sous-séquence de la procédure de chiffrement. Même si le terme "simple" est utilisé, cela ne signifie pas qu'une seule trace de puissance est nécessaire. De multiples traces pourraient être utilisées pour réduire le rapport signal/bruit entre le signal de fuite réel et le bruit électronique ou de mesure. Cependant, le terme "simple" fait référence au fait qu'au cours d'une SPA, la relation entre les différentes traces et les variations des valeurs intermédiaires qui produisent la trace de fuite enregistrée n'est pas exploitée.

Un exemple classique d'une attaque SPA est la reconnaissance des motifs de fuite qui sont causés par les opérations individuelles d'un processeur. Par exemple, dans les implémentations non sécurisées de l'algorithme de multiplication par un scalaire d'un point d'une courbe elliptique, ou dans la multiplication de nombres premiers dans le chiffrement RSA, le profil de fuite diffère selon qu'il s'agit d'une simple opération de mise au carré ou d'une opération de mise au carré suivie d'une multiplication. Ces motifs peuvent même être observés dans une simple inspection visuelle d'une trace de puissance, permettant d'extraire directement de la trace de puissance l'exposant utilisé pour l'exponentiation (qui peut être la clé elle-même ou liée à la clé).

Attaques non supervisé

En plus des variations de puissance, il existe des effets corrélés aux valeurs des données manipulées. Ces variations ont tendance à être plus petites et sont parfois éclipsées par des erreurs de mesure et d'autres bruits. Dans ces cas, il est souvent encore possible d'exploiter

les fuites en utilisant des fonctions statistiques adaptées à l'algorithme cible.

Une attaque non supervisée classique peut être divisée en trois phases : la phase de collecte des fuites, la phase de construction des hypothèses et la phase de correspondance des hypothèses.

Phase de collecte des fuites : Pendant cette phase, les traces de fuite sont collectées sous différentes entrées, par exemple différents textes en clair qui sont chiffrés à l'aide de la même clé secrète. L'exploitation des différences dans les traces de fuite nécessite un alignement aussi exact que possible des traces individuelles les unes par rapport aux autres. Dans les attaques pratiques, il n'existe souvent pas de point de référence dans le temps pour chaque chiffrement. Cette phase pourrait donc comporter une phase de post-traitement au cours de laquelle les traces de fuite sont d'abord alignées.

Phase de construction des hypothèses : Cette phase est réalisée en l'absence de connaissance de la clé secrète utilisée, l'étape suivante consiste donc à générer des hypothèses basées sur la clé secrète inconnue. Il est souvent nécessaire d'utiliser une approche diviser pour mieux régner, car la construction d'hypothèses basées sur un espace clé de 128 bits, comme pour un AES-128 complet, serait trop complexe. Par conséquent, il faut d'abord trouver un point approprié dans l'algorithme attaqué où une relation peut être établie avec la clé secrète et les données d'entrée connues. À ce moment-là, la clé et les données sont traitées en paquets de taille inférieure à 128 bits. Par exemple, un point approprié pour la construction de cette hypothèse serait le résultat de l'addition de la clé et du texte en clair dans le premier tour d'un AES, suivi de la recherche de la S-box sur 8 bits. On peut ainsi calculer des hypothèses individuellement pour tous les morceaux de 8 bits de la clé de 128 bits.

Phase de correspondance des hypothèses : Le modèle de puissance hypothétique pour les principales suppositions est évalué statistiquement par rapport aux observations réelles dans les traces de fuite. Il existe de nombreux outils mathématiques qui permettent de distinguer les clés utilisées sur les fuites par canaux auxiliaires pour sélectionner le candidat clé le plus probable parmi l'ensemble des clés hypothétiques. Ces distinctions sont basées sur différentes méthodes statistiques, comme la corrélation de Pearson, la différence des moyennes ou l'analyse d'informations mutuelles. L'objectif de tous ces outils est de trouver et de quantifier les dépendances entre le modèle de fuite hypothétique afin de déterminer la clé utilisée.

Attaques supervisé

En considérant les attaques supervisées, nous partons du principe qu'un dispositif exécute l'une des K séquences d'opérations possibles, désignées par $O_1, \ldots, O_k$. Cela peut correspondre, par exemple, à l'exécution du même code avec différentes valeurs de bits de clés. L'adversaire, capable d'échantillonner le canal auxiliaire pendant cette opération, vise à identifier l'opération effectuée ou à réduire significativement le nombre d'hypothèses possibles pour cette opération.

Dans le domaine du traitement du signal, il est courant de modéliser l'échantillon observé comme une combinaison d'un signal intrinsèque (généré par l'opération) et de bruit (soit intrinsèque, soit ambiant). Alors que le signal reste constant pour les invocations répétées de l'opération, le bruit est mieux modélisé comme un échantillon aléatoire tiré d'une distribution de probabilité qui dépend des conditions de fonctionnement et d'autres conditions ambiantes.

Pour déterminer l'hypothèse la plus probable à partir d'un petit ensemble d'échantillons S, l'adversaire peut utiliser l'approche du maximum de vraisemblance. Celle-ci consiste à choisir l'opération qui maximise la correspondance entre les traces observées et le modèle de l'opération. Afin de calculer cette correspondance, l'adversaire doit modéliser avec précision à la fois le signal intrinsèque et la distribution de probabilité du bruit pour chaque opération.

L'adversaire utilise un dispositif expérimental, identique au système analysé, pour cibler une petite portion de l'échantillon S dépendant uniquement de quelques bits de clés non connus. Par le biais d'une expérimentation méthodique, il génère des modèles correspondant à chaque possible valeur des bits clés non connus. Chaque modèle est constitué des distributions de probabilité moyennes du signal et du bruit. Ces modèles servent ensuite à classifier cette partie de S et à limiter les choix pour les bits clés à un ensemble restreint. Cette procédure est réitérée avec un préfixe de S de plus en plus long impliquant un nombre croissant de bits clés. Seul un nombre réduit de possibilités pour la partie de la clé examinée jusqu'à présent est conservé. De la sorte, les attaques par template appliquent principalement une stratégie d'extension et de réduction pilotée par l'échantillon unique S ciblé : elles utilisent des préfixes de S de longueur croissante et les modèles correspondants pour restreindre l'espace des clés possibles. Le succès de cette approche dépend essentiellement de l'efficacité de la stratégie de réduction pour contenir l'explosion combinatoire dans le processus d'extension.

Les attaques par template sont particulièrement efficaces sur les implémentations d'algorithmes cryptographiques sur des dispositifs CMOS, du fait de leur contamination et de leur diffusion sur plusieurs cycles dans une section de calcul. Dans les dispositifs CMOS, la manipulation directe des bits clés induit une infiltration dans l'état du dispositif et ces fuites d'état peuvent perdurer pendant plusieurs cycles. De surcroît, d'autres variables influencées par la clé, telles que les indices et les valeurs des tables dépendantes de la clé, engendrent une contamination additionnelle lors de cycles ultérieurs. L'ampleur de la contamination gouverne la réussite de la réduction des nouveaux bits clés introduits lors de la phase d'expansion. Il faut s'attendre à ce que si deux clés sont presque identiques, la réduction ne parvienne pas à éliminer l'une d'entre elles, même en tenant compte des effets de la contamination. La diffusion est une propriété cryptographique bien établie selon laquelle de petites différences dans les bits clés sont amplifiées dans les parties ultérieures du calcul. Ainsi, même si certains candidats pour les bits de clés n'ont pas été éliminés en raison des effets de la contamination, la diffusion garantira que les clés très proches seront rapidement éliminées.

L'implémentation d'un algorithme sur un dispositif spécifique impose des limites théoriques à la réussite de l'attaque par template. Pour se rapprocher au mieux de cette limite théorique, l'adversaire doit disposer de caractérisations extrêmement précises et robustes du bruit. Même si ces caractérisations peuvent être hautement sophistiquées, en pratique, des approximations comme un modèle gaussien multivarié pour les distributions du bruit produisent des résultats satisfaisants. De plus, les avancées dans le domaine du machine learning permettent des attaques supervisées encore plus précises, à condition que des données d'entraînement suffisantes soient disponibles.

2.1.4 Méthode de test

Pour évaluer la résistance des implémentations face aux attaques par canaux auxiliaires, deux méthodologies de test sont principalement utilisées. La première, dite empirique, se base sur l'analyse des fuites et des méthodes statistiques pour évaluer la résistance. La seconde, dite formelle, vérifie si la conception respecte certaines propriétés mathématiques. Ces deux méthodes sont souvent utilisées de manière complémentaire : on vérifie d'abord formellement la résistance de l'implémentation, puis on procède à une vérification empirique pour confirmer cette résistance.

Méthode empirique

Analyser la résistance contre les attaques par canaux auxiliaires des implémentations de manière empirique, est souvent effectué par l'intermédiaire d'un t-test selon la méthode de Goodwill et al. [Goo+11] . Nous notons que les t-tests ne sont pas adaptés pour prouver des déclarations générales sur la sécurité d'un modèle (pour toutes les conditions et tous les temps de signaux possibles) comme il serait nécessaire pour une vérification complète de la sécurité. Les t-tests n'autorisent des déclarations que pour les dispositifs testés et dans les limites du dispositif de mesure. De nombreux ouvrages testent les circuits masqués sur un FPGA et effectuent le t-test sur les traces recueillies à partir des mesures de puissance. Cette approche présente l'inconvénient qu'en raison des niveaux de bruit relativement élevés, l'évaluation est généralement limitée à des t-tests multivariés du premier et du deuxième ordre. Cependant, dans la pratique, les t-tests se sont avérés très sensibles et utiles pour tester la résistance des canaux auxiliaires de circuit. Les traces de signaux sont enregistrées lors des simulations post-synthèse des netlists, qui sont exemptes de bruit et nous permettent d'évaluer les conceptions jusqu'au troisième ordre. L'utilisation de traces de fuites post-synthèse sur des traces collectées à partir d'une conception de FPGA ou d'ASIC montre quelques différences qui sont dans certains cas très bénéfiques, mais il y a aussi des inconvénients. Tout d'abord, les traces de fuite après synthèse sont totalement exemptes de bruit environnemental et de variations des conditions de fonctionnement comme la température ou la tension d'alimentation. En conséquence, les violations de la sécurité d'ordre D sont constatées avec beaucoup moins de traces de fuite. Un autre grand avantage est que les t-tests peuvent être effectués soit à un niveau assez grossier, en prenant en compte tous les signaux ensemble, soit à une granularité très fine en utilisant des signaux individuels. Cette dernière méthode permet de localiser directement la source de la fuite au niveau du signal, ce qui facilite grandement la conception des tests. L'un des inconvénients de cette approche est que les traces post-synthèse n'utilisent pas une source de fuite réelle existante. Cependant, un t-test effectué sur une puce ASIC ou sur la conception d'un FPGA ne permet de donner qu'une fuite existante sur ce dispositif et ne donne même pas de garantie sur son comportement à l'avenir, car les retards de signaux peuvent changer dans les conditions environnementales et au cours du cycle de vie d'un dispositif. Dans le cas de la netlist synthétisée simulée, les retards de signaux sont basés sur les retards de la porte unifié qui entraînent également des glitchs de signaux qui apparaissent à partir des portes logiques en cascade. Les glitchs qui pourraient résulter des longueurs de fils de différentes, et autres effets parasites, ne sont cependant pas modélisés et sont

donc plus susceptibles d'apparaître sur les t-tests basés sur FPGA ou ASIC. Pour vérifier qu'un appareil ne présente aucune fuite exploitable, deux séries de traces sont collectées par t-test :

1. un ensemble avec des entrées choisies au hasard
2. un ensemble avec des entrées contenant des données sensibles

La valeur t est calculée selon l'équation ci-dessous où X désigne la moyenne de l'ensemble de traces respectif, S2 est la variance et N est la taille de l'ensemble.

$$t = \frac{X_1 - X_2}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}} \tag{2.1}$$

L'hypothèse nulle est que les moyennes des deux ensembles de traces sont égales, ce qui est accepté si la valeur t calculée est inférieure au seuil de $\pm$4,5. Si la valeur t dépasse ce seuil, alors l'hypothèse nulle est rejetée avec une confiance supérieure à 99,999 % pour les ensembles de traces suffisamment importants. Une étape dite de prétraitement centré du produit, avec des points de trace à l'intérieur d'une fenêtre de six cycles, est effectuée pour les t-tests d'ordre supérieur. Au-delà de cette fenêtre de temps, on s'assure que les produits intermédiaires n'ont aucun lien avec les entrées. Nous combinons donc plusieurs points de trace en normalisant d'abord les moyennes des points de trace, puis en multipliant les valeurs résultantes par d'autres points normalisés à l'intérieur de la fenêtre temporelle.

Méthode formelle

Le masquage booléen d'ordre d (également appelé d-share) d'une variable $a \in GF(2^m)$ est défini par l'ensemble $s_a = \{a_i\}_{i=1}^n$, où chaque élément $a_i \in GF(2^m)$ est généré aléatoirement. Les données masquées sont obtenues en combinant ces différentes parts. Pour récupérer la donnée originale a, il est nécessaire d'exécuter l'opération inverse sur les d-shares.

Le nombre de parts n est dicté par le schéma de masquage et est toujours supérieur à l'ordre de sécurité d. Aucune combinaison de d parts ou moins ne doit révéler d'information sur a.

Toute fonction $F(a) = x$ peut être implémentée en utilisant une combinaison d'opérations affines et de multiplications dans le champ donné. Le calcul masqué d'une fonction affine A

est trivial, car la fonction peut simplement être appliquée à chaque part individuellement : $A(a_i) = x_i$.

Le calcul du ET logique masqué, en revanche, est plus complexe. Un exemple de partage pour la fonction $F(a, b) = ab$ qui utilise trois parts et une variable aléatoire r_i est donné par [ISW03a] comme suit :

$$
\begin{aligned}
t_1 &= (a_1 b_2 \oplus r_1) \oplus a_2 b_1, \\
t_2 &= (a_1 b_3 \oplus r_2) \oplus a_3 b_1, \\
t_3 &= (a_2 b_3 \oplus r_3) \oplus a_3 b_2, \\
x_1 &= a_1 b_1 \oplus r_1 \oplus r_2, \\
x_2 &= a_2 b_2 \oplus t_1 \oplus r_3, \\
x_3 &= a_3 b_3 \oplus t_2 \oplus t_3.
\end{aligned}
$$

Definition 2.1.1 (Sécurité d-probing [ISW03a])**.** Un circuit est dit d-probing sécurisé si et seulement si chaque d-tuple de ses variables intermédiaires est indépendant de toute variable sensible.

Il a été démontré que dans un circuit exempt de glitches, la sécurité d-probing implique la sécurité du modèle bruité, qui suppose que chaque opération fuit indépendamment. Cependant, ce modèle présente deux inconvénients principaux :

1. La sécurité contre les attaques par canaux auxiliaires sur un composant individuel ne garantit pas la sécurité lorsque ces composants sont combinés de manière arbitraire.
2. L'hypothèse d'un circuit sans glitchs a été réfutée comme étant irréaliste, en particulier dans les implémentations matérielles.

Pour pallier ces lacunes, de nouveaux modèles ont été introduits :

Definition 2.1.2 (Gadget composable)**.** Un gadget d-probing sécurisé est composable si la combinaison arbitraire de tels gadgets résulte en un circuit d-probing sécurisé.

Des travaux ultérieurs ont montré que tout gadget qui satisfait la propriété d-SNI est composable [Bar+16a]. En outre, tout circuit composé de gadgets affines et d-SNI est d-probing sécurisé.

Definition 2.1.3 (Non-Interférence Forte d-SNI [Bar+16a])**.** Un gadget est dit d-SNI si, pour tout ensemble p_1 de sondes sur ses valeurs intermédiaires et tout ensemble p_2 de

sondes sur ses sorties, l'ensemble total de sondes peut être simulé avec $p_1 + p_2 \leq d$, où p_1 est le nombre de parts de chaque entrée.

En somme, même si un circuit est d-probing sécurisé, cette sécurité théorique peut ne pas se traduire en sécurité pratique en raison de facteurs tels que les glitchs. Pour remédier à ces défauts, le modèle de d-probing étendu a été créé [MRB18]. Ce modèle est un cadre théorique solide qui peut être utilisé avec un haut degré de confiance pour évaluer la sécurité des implémentations cryptographiques dans un environnement sujet aux glitchs et autres anomalies matérielles.

Definition 2.1.4 ([MBR19])**.** Considérons un gadget avec $d + 1$ parts d'entrée a_i, où $i \in \{1, \ldots, d+1\}$. Soit O un ensemble d'observations d'au plus d sondes glitch-extended dans $GF(2^m)$. Le gadget est dit d-GSNI si la condition suivante est remplie :

$$\exists P \subset \{1, \ldots, d+1\} \text{ et son complément } \overline{P} \text{ avec } |P| = p_1 \text{ telle que } I(O; a_{P'}|a_P) = 0.$$

Cette dernière définition repose sur les principes de la théorie de l'information et est en concordance avec la Définition 2.1.1 si des sondes conventionnelles sont utilisées au lieu des sondes glitch-extended.

2.1.5 Classifications des contremesures

Depuis la mise en évidence de la première attaque par canaux auxiliaires en 1999 [KJJ99], un éventail de contre-mesures a été élaboré. Ces mécanismes de défense peuvent être méthodiquement classifiés en plusieurs catégories distinctes :

- Intégration de fluctuations temporelles : cette stratégie vise à contrer les attaques fondées sur l'analyse de consommation électrique, de synchronisation, ou des émissions électromagnétiques.
  - Modification du rapport cyclique ou de la fréquence d'oscillation.
  - Recours à des circuits asynchrones, indépendants d'une horloge externe pour le séquencement des opérations.
  - Insertion d'instructions inopérantes générées aléatoirement, n'ayant aucune conséquence sur la fonctionnalité intrinsèque du circuit.
- Utilisation de l'homomorphisme cryptographique : Exploitation des propriétés arith-

métiques afin de diversifier les chemins de calcul et obfusquer le processus d'exécution.

- Application de la randomisation : cette technique vise à modifier la représentation des informations sensibles, les rendant inaccessibles directement par des entités malintentionnées.
  - Technique de masquage : fusion de l'information sensible avec une valeur aléatoire pendant le processus de calcul. À la conclusion de cette opération, une démarche de démystification est entreprise pour obtenir le résultat escompté.
  - Insertion d'interférences dans la consommation électrique, que ce soit à l'échelle macroscopique du circuit ou à un niveau plus granulaire.
    * Préallocation de valeurs dans les registres ou sur les voies de transmission de données.
    * Fluctuation contrôlée de la tension d'alimentation.
- Détecteurs d'intrusion : ces dispositifs ont la capacité d'identifier des tentatives d'attaque, telles que la dépackageisation, l'utilisation de sondes électromagnétiques ou les manipulations d'horloge. Il est néanmoins crucial de noter que leur efficacité peut être contournée par des techniques avancées.
- Atténuation des émissions par canaux auxiliaires : cette stratégie consiste à minimiser les corrélations au niveau des composants logiques entre les données sensibles et les émissions parasitaires. Diverses solutions ont été avancées pour équilibrer ou réduire la consommation électrique, parmi lesquelles on peut citer : Dual-Rail Random Switching Logic (DRSL) [CZ06], Masked Dual-Rail Pre-charge Logic (MDPL) [PM05], Dual-Rail Pre-charge Logic (TDPL) [Buc+06], Random Switching Logic (RSL) [SSI04], Sense Amplifier Based Logic (SABL), ainsi que Wave Dynamic Differential Logic (WDDL).

2.2 Attaque par injection de fautes

L'examen des fuites par canaux auxiliaires d'un circuit numérique constitue une étape préalable à l'ingérence directe dans les opérations de ce dernier. L'optimum fonctionnel d'un circuit est prescrit en fonction de paramètres physiques spécifiques tels que la température, la tension d'alimentation, et l'hygrométrie. Dès les années 1970, concomitamment

avec l'avènement de l'ère spatiale, des anomalies dans le fonctionnement des circuits électroniques ont été observées, attribuées aux radiations cosmiques au-delà de la stratosphère terrestre. Avec la miniaturisation des transistors et la diminution conséquente de leur énergie d'activation, de tels phénomènes ont également été constatés dans des environnements terrestres et aéronautiques, bien que dans des proportions moindres. Ces anomalies sont communément désignées sous le terme de *soft errors*.

Outre ces fautes induites naturellement, existent des perturbations intentionnelles, connues sous le nom d'attaques par injection de fautes. Ces dernières visent à altérer le comportement du circuit de manière délibérée et contrôlée en modifiant ses conditions opératoires, parfois de façon abrupte. Elles peuvent être de nature invasive, requérant des modifications matérielles pour leur exécution, ou semi-invasive, sans altération physique mais entrainant néanmoins des modifications fonctionnelles. Il convient de souligner que, par leur essence même, ces attaques sont considérées comme invasives, contrastant avec les attaques par canaux auxiliaires qui sont essentiellement non invasives. Les conséquences de telles intrusions peuvent être graves : déni de service, modification du comportement du système à des fins malveillantes, ou encore la divulgation d'informations confidentielles, telles que des clés cryptographiques.

La problématique des attaques par injection de fautes s'est accentuée vers la fin des années 1990 [BS97], bien que les *soft errors* dans le contexte spatial aient été documentés depuis les années 1970. Il est à noter que, contrairement aux fautes spatiales qui sont largement étudiées, l'accent dans le domaine de l'injection de fautes a principalement été mis sur les crypto-processeurs, susceptibles de nombreux modes d'exploitation de ces vulnérabilités.

Enfin, il est impératif de distinguer entre la sécurité et la sûreté. La première implique la présence d'un adversaire malveillant, alors que la deuxième se caractérise par un risque inhérent, de nature statique et probabiliste. La sûreté se concentre principalement sur la correction des erreurs, offrant ainsi un plus grand éventail de possibilités pour un attaquant potentiel. En revanche, la sécurité est davantage centrée sur la détection des anomalies et la mise en œuvre de routines spécifiques pour la gestion des incidents. Ces deux paradigmes, bien que distincts dans leur approche de la gestion des risques, présentent des défis significatifs lorsqu'il s'agit de les intégrer de manière cohérente au sein d'un même système.

2.2.1 Fautes stochastiques

Divers types de fautes peuvent survenir, indépendamment de la méthode d'injection de fautes utilisée. De nombreux effets liés à un événement unique (SEEs, pour Single-Event Effects) de nature transitoire sont possibles :

- Les transitoires d'événements uniques (SETs, pour Single-Event Transients) entraînent une modification temporaire de la tension en sortie d'une porte logique.
- Les bouleversements d'événements uniques (SEUs, pour Single-Event Upsets) provoquent une inversion de la valeur stockée dans une cellule de mémoire.
- Les interruptions fonctionnelles d'événements uniques (SEFIs, pour Single-Event Functional Interruptions) induisent une dysfonction nécessitant un redémarrage du système [Kog+97].

Dans les cas où l'énergie concentrée est excessivement élevée, les SEEs peuvent entraîner des dommages irréversibles :

- Les accrochages d'événements uniques (SELs, pour Single-Event Latchups) activent en permanence un thyristor parasite dans le CMOS.
- Les grillages d'événements uniques (SEBs, pour Single-Event Burnouts) entraînent une polarisation directe dans le transistor parasite d'un power MOSFET.
- Les ruptures de grille d'événements uniques (SEGRs, pour Single-Event Gate Ruptures) induisent un champ électrique transitoire à travers l'oxyde de grille d'un power MOSFET.

Il est également possible que plusieurs de ces fautes se produisent simultanément, on parle alors d'effets liés à des événements multiples (MEEs, pour Multiple-Event Effects)voir figure 2.2. En outre, avec la réduction des tailles de gravure, une particule peut provoquer plusieurs soft errors[Pag+11], [ZES89]. En conséquence, une soft error peut affecter plusieurs bits d'une même variable.

Selon des données disponibles dans la littérature scientifique, notamment celles de la NASA, un processeur serait en moyenne soumis à 5-10 fautes par heure. Plus de 99,99% de ces fautes seraient de nature transitoire et 99,9% seraient des SEUs [Spr01].

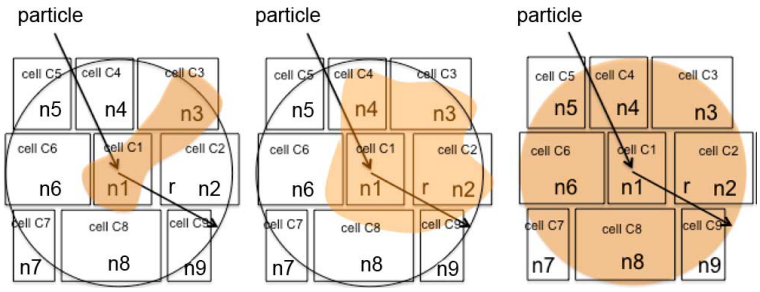

FIGURE 2.2 : Influence d'une faute.

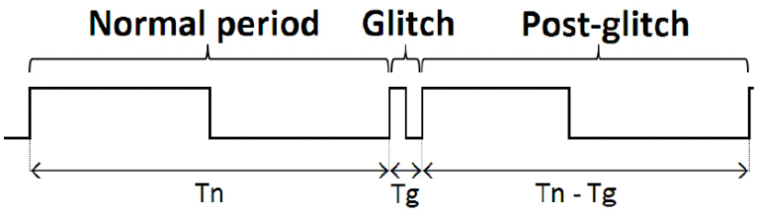

FIGURE 2.3 : Illustration d'un glitch sur le signal d'horloge.

2.2.2 Faute non stochastique

Glitch d'horloge

Le phénomène de "glitch d'horloge" est illustré par la Figure 2.4, qui met en lumière les conséquences des violations des contraintes temporelles sur le fonctionnement d'un circuit électronique. Ce phénomène est initié par une altération de la fréquence nominale de l'horloge, voir figure 2.3, T_n via l'insertion d'un glitch de durée T_g. Le principe repose sur l'accélération temporaire de la fréquence d'horloge, formalisée par $T_g \ll T_n$, pouvant entraîner des violations de contraintes temporelles. Notons que si $T_g \ll T_n$, le comportement post-glitch du circuit demeure inchangé [BGV11].

Les erreurs résultant des violations de contraintes temporelles présentent deux caractéristiques distinctives :

- Les fautes sont fonction des données en cours de traitement. Plus précisément, ces données peuvent influencer les chemins critiques, en particulier dans les architectures de processeurs d'application.
- Le mécanisme d'injection peut conduire à des états métastables lorsque le niveau de stress imposé au circuit est insuffisant. Ces états peuvent cependant devenir déterministes si le stress augmente, d'où la nécessité d'optimiser les paramètres d'injection pour induire des fautes.

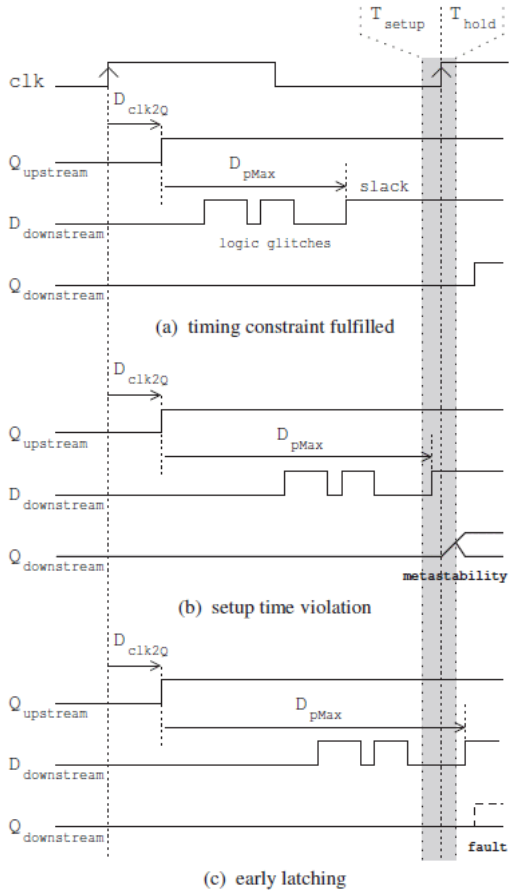

FIGURE 2.4 : Schéma d'une violation de contrainte temporelle.

Ces propriétés sont communes à toutes les formes d'injections perturbant les contraintes temporelles du circuit, incluant les glitchs d'alimentation, les variations de température et les injections électromagnétiques (EM).

Glitch d'alimentation

Dans le cas de glitch d'alimentation, il s'agit de moduler brièvement la tension d'alimentation du circuit, souvent en la réduisant. L'explication la plus plausible pour les erreurs résultantes est une modification des propriétés physiques des transistors, induisant des erreurs dans les temps de SETUP et de HOLD [DOCB06]. Bien que la causalité exacte demeure incertaine, la violation des timings semble être l'hypothèse la plus plausible [Zus+13]. De plus, l'article en question établit une équivalence entre les glitchs d'alimentation et ceux de l'horloge.

Il est à noter que la probabilité d'occurrence des fautes peut être accrue en utilisant des signaux d'alimentation de forme aléatoire plutôt que des signaux carrés [BFP19]. Cette méthode augmente la probabilité de fautes de 2 à 10 fois par rapport à l'injection de signaux carrés.

Changement de température

Les dispositifs électroniques fonctionnent dans des plages de température spécifiques. Bien que de nombreux travaux aient évoqué la possibilité d'attaques basées sur des variations de température, peu ont mis ces concepts en pratique. Le refroidissement est couramment employé pour récupérer des informations résiduelles dans les mémoires SRAM après extinction du système [Hal+09]. En ce qui concerne l'injection de fautes, l'accent est généralement mis sur l'élévation de la température. Par exemple, il a été démontré qu'une lampe de bureau de 50W peut induire des erreurs avec une probabilité de 71,4% en chauffant un dispositif à 100 °C, avant que celui-ci ne cesse de fonctionner [HS13b]. Une exposition prolongée à des températures élevées peut entraîner une instabilité affectant jusqu'à 30% des bits de la mémoire.

Il est important de noter que les variations de température ont un impact sur l'ensemble du circuit électronique et non sur des composants spécifiques. De plus, la gestion temporelle de ces variations n'est généralement pas précise. Enfin, les erreurs induites par ces méthodes peuvent être irréversibles, ce qui limite considérablement l'utilité de telles attaques.

Injection combinée

Il est envisageable de fusionner diverses techniques d'injection de fautes, notamment les glitches temporels, les variations d'alimentation électrique et les changements de température [KJP14]. L'avantage de cette approche combinée réside dans la possibilité d'ajuster finement les paramètres physiques, ce qui améliore la précision des méthodes d'injection de fautes. Cette synergie permet de cibler des zones spécifiques du circuit de manière plus efficace et économique. Toutefois, il est important de souligner que ces techniques combinées n'engendrent pas de nouveaux types de fautes. Elles servent plutôt à augmenter la reproductibilité des fautes déjà identifiées.

Injection laser

Le laser (Light Amplification by Stimulated Emission of Radiation) est une source de lumière électromagnétique monochromatique, unidirectionnelle et cohérente. Avec un diamètre pouvant être de l'ordre du micromètre et la capacité de traverser divers matériaux en un temps très court, le laser est particulièrement bien adapté pour l'injection de fautes dans les circuits électroniques. La première utilisation documentée de cette technique remonte à 2003 [SA03], et une explication détaillée de son efficacité a été fournie par JM Dutertre [Dut+18].

Lors de la planification d'une attaque par laser, plusieurs paramètres doivent être pris en compte : le diamètre du faisceau, la longueur d'onde, les coordonnées ciblées, la quantité d'énergie délivrée et la durée d'exposition. Une attention particulière doit être accordée à cette dernière ainsi qu'à la quantité d'énergie, car une exposition excessive peut entraîner des erreurs permanentes [Dar+01]. Les avantages de cette méthode incluent la haute reproductibilité des fautes induites et une précision exceptionnelle, pouvant aller jusqu'à l'octet ou même le bit, même sur des technologies de pointe [Ago+10] [Sel+16]. De plus, des bancs d'injection laser multispots sont désormais disponibles sur le marché, offrant des configurations à 2 et 4 spots[Col+21]. Il est à noter que bien que le diamètre du laser soit très petit, sa précision est encore plus grande en raison de la concentration d'énergie au centre du faisceau [God+09].

L'injection de fautes par laser peut être réalisée soit sur la face avant, soit sur la face arrière du circuit, chacune ayant ses propres caractéristiques tout en conservant le même principe de base. Pour les attaques sur la face avant, le positionnement est facilité par la visibilité des composants, mais la présence de couches métalliques réfléchissantes peut réduire la

Energie[nJ]	0.4	0.5	0.8	1	1.5	2	3	4	5
nb de fautes	1	8	21	23	24	24	26	30	31
nb de fautes 1-bit	1	8	15	17	10	7	7	9	9
nb de fautes 2-bit	-	-	6	6	7	5	4	5	6
nb de fautes 3-bit	-	-	-	-	4	7	8	4	4
nb de fautes 4-bit	-	-	-	-	3	3	3	5	1
nb de fautes 5-bit	-	-	-	-	-	1	1	2	4
nb de fautes 6-bit	-	-	-	-	-	1	1	2	2
nb de fautes 7-bit	-	-	-	-	-	-	1	2	4
nb de fautes 8-bit	-	-	-	-	-	-	-	1	1

Table 2.2 : tableau récapitulatif du nombre et taille de faute en fonction de la puissance du laser.

précision. Les longueurs d'onde d'environ 523 nm sont couramment utilisées dans ce cas. Pour les attaques sur la face arrière, des longueurs d'onde d'environ 1064 nm sont préférées pour une meilleure pénétration à travers le silicium. Bien que le positionnement soit plus complexe, la précision est généralement supérieure car on évite la couche métallique. Les coefficients d'absorption et de réflexion du silicium, qui varient en fonction de la longueur d'onde, doivent également être pris en compte [BJ15].

Les résultats d'une série d'injections de fautes sur une puce de 28 nm sont présentés dans le tableau 2.2. Les données présentées dans ce tableau mettent en évidence la capacité des fautes à affecter un grand nombre de bits consécutifs. Cela signifie qu'une seule source de faute peut entraîner plusieurs erreurs simultanément. En augmentant la puissance du laser, il est possible d'induire un grand nombre de fautes consécutives sur une donnée spécifique.

Injection EM

Dès 2002, Quisquater et Samyde ont démontré que le champ magnétique généré par une sonde électromagnétique pouvait perturber les opérations d'un circuit électronique [QS02]. Ce n'est qu'en 2007 que cette technique a été exploitée pour mener une attaque, spécifiquement l'attaque Bellcore [SH07]. Contrairement à une idée reçue largement répandue, les fautes induites par l'injection électromagnétique (EM) ne se limitent pas uniquement à des erreurs de synchronisation [Deh+12]. Elles peuvent également provoquer des "samplings faults", c'est-à-dire des erreurs au niveau des portes logiques plutôt qu'au niveau du circuit global (comme les bits-set et les bits-reset) [DLM19] [OGSM17].

Un avantage majeur de l'injection de fautes par électromagnétisme est qu'elle peut être

réalisée sans avoir à décapsuler le circuit. Toutefois, dans la pratique, il est souvent bénéfique de décapsuler le circuit pour améliorer l'efficacité de l'injection. Quant à la précision temporelle de cette méthode, elle demeure inférieure à celle des injections par glitch ou par laser, car il est nécessaire de précharger le générateur d'impulsions avant de lancer l'attaque.

2.2.3 Méthode de test

Les méthodes de test pour évaluer la résilience des systèmes électroniques face aux fautes peuvent être classées en quatre grandes catégories, en fonction du niveau d'abstraction auquel elles opèrent :

- Tests sur le système physique
- Tests logiciels
- Simulation
- Émulation

Chacune de ces méthodes peut être appliquée à différents niveaux d'abstraction, allant de l'assembleur et des jeux d'instructions, au niveau Register-Transfer Level (RTL), en passant par les portes logiques, jusqu'à la post-implémentation.

Nous n'aborderons pas ici en détail les méthodes d'injection physique de fautes, car elles ont déjà été discutées dans la partie I de ce document. Il est toutefois important de noter que dans le contexte de la sûreté des systèmes, des outils ont été développés pour automatiser les campagnes de tests d'injection de fautes [Arl+90], [Mad+94].

Cette synthèse s'appuie sur diverses revues de littérature existantes, dont j'ai tenté de réconcilier les contributions respectives. Elle offre également un examen plus approfondi des différents outils disponibles pour chaque catégorie de méthodes de test [Esl+20], [KDN14].

Niveau d'abstraction

La résilience aux fautes peut être testée à plusieurs niveaux d'abstraction, chacun ayant ses avantages et inconvénients en fonction du contexte. Voici une exploration des différents niveaux, en commençant par les plus abstraits :

- Le niveau d'assembleur ou de jeux d'instructions est le plus abstrait. Il implique la modification du code source pour altérer le comportement du processeur. Ce niveau permet de simuler des fautes tout en tenant compte des spécificités architecturales. Son avantage réside dans la facilité d'injection de fautes dans des applications et des systèmes d'exploitation. Toutefois, cette méthode est limitée aux ressources accessibles via le jeu d'instructions et modifie le code source, ce qui signifie que le code testé diffère du code en fonctionnement nominal.
- Au niveau transactionnel, la simulation se concentre sur le comportement des composants en termes de transactions. On peut ainsi simuler des fautes dans les registres, les mémoires et les transactions sur le bus. Cette méthode est facile à mettre en œuvre et permet également de simuler des erreurs plus spécifiques liées à l'implémentation, comme la corruption de registres ou de mémoire.
- Le niveau Register-Transfer Level (RTL) offre une vue réaliste de l'implémentation et des signaux du circuit. Cependant, cette méthode est surtout utile pour une analyse fonctionnelle et ne prend pas en compte les aspects temporels ou de consommation d'énergie.
- Au-delà du RTL, il y a le niveau de la synthèse. Les tests à ce niveau sont assez généralisables à différentes technologies, car la synthèse est une étape commune à toutes les implémentations matérielles. L'avantage est cette généralisabilité, mais elle est aussi un inconvénient car il est difficile d'obtenir des informations précises sur le timing et la consommation d'énergie.
- Les simulations post-placement et routage offrent un modèle très proche du circuit physique, y compris des valeurs réalistes de timing et de consommation. Cependant, ces simulations sont très coûteuses en temps et en ressources.
- Enfin, il est possible de tester directement sur un circuit physique, qu'il soit un ASIC ou un FPGA. Cette approche réduit considérablement le temps nécessaire pour les campagnes d'injection de fautes et offre des mesures de timing et de consommation réalistes. Néanmoins, l'accès aux interfaces et à l'état interne du circuit est limité, nécessitant une instrumentation substantielle. De plus, les résultats obtenus sur une technologie (ASIC ou FPGA) ne sont pas directement transférables à l'autre en raison de leurs différences fondamentales.

Chaque niveau d'abstraction présente ses propres avantages et défis, et le choix du niveau

approprié dépendra des objectifs spécifiques.

Injection Software

L'injection logicielle de fautes permet d'introduire des erreurs dans un système pendant son fonctionnement nominal. Cette méthode peut être appliquée dans divers contextes, tels que la simulation, l'émulation ou le fonctionnement sur un système physique. Elle utilise les composants et les fonctionnalités intrinsèques du système, comme le JTAG, le code source ou les mécanismes de débogage, pour effectuer les injections. Typiquement, cette technique modifie l'état des mémoires, des registres, ou même le résultat d'un calcul.

L'injection logicielle peut être réalisée de manière statique ou dynamique :

- L'injection statique de fautes intervient lors de la phase de compilation. Le système est modifié à ce stade pour simuler une faute. Cette méthode s'intègre facilement dans un flux de conception, car elle n'affecte que la compilation et est donc indépendante du système physique.
- L'injection dynamique de fautes, en revanche, ajoute un déclencheur juste avant l'instruction qui sera altérée. Cette approche utilise les mécanismes d'exception et d'interruption du système. Contrairement à l'approche statique, elle n'altère pas les instructions existantes mais en ajoute de nouvelles.

L'un des avantages majeurs de cette technique est qu'elle ne nécessite pas de matériel spécifique pour la campagne d'injection de fautes. Elle utilise uniquement les fonctionnalités disponibles sur la cible. Étant donné qu'elle opère au niveau de l'assembleur, elle est principalement utilisée pour caractériser les fautes au niveau de l'application et du système d'exploitation. De plus, la vitesse d'exécution pendant l'injection est presque identique à celle du fonctionnement nominal, ce qui permet de réaliser des campagnes d'injection à grande échelle. Toutefois, les modifications apportées au système peuvent introduire des erreurs, notamment en ce qui concerne le temps d'exécution.

Injection par simulation

L'injection de fautes par simulation nécessite la création d'un modèle de simulation, qui peut être conçu à divers niveaux d'abstraction, tels que le jeu d'instructions, le niveau transactionnel, le Register-Transfer Level (RTL), la post-synthèse, ou encore le post-placement/routage. Dans l'état actuel de la recherche, les outils d'injection de fautes par

simulation sont majoritairement basés sur un modèle RTL [STB97]. Chaque faute introduite dans une campagne d'injection correspond à une simulation unique.

Deux approches principales sont utilisées pour mener une campagne d'injection : la modification du code source et l'utilisation des outils de simulation. Dans la première approche, le code VHDL peut être modifié pour créer des "mutants", qui sont des versions du composant conçues pour exécuter une faute spécifique lors de l'exécution. Une autre méthode consiste à ajouter des "saboteurs", qui sont des composants spécialement conçus pour injecter des fautes lors de l'exécution (voir Figure 2.5).

Un mutant est une variante du code source d'un composant, spécifiquement modifiée pour exécuter une faute donnée lors de son fonctionnement. L'un des principaux avantages de cette méthode est sa flexibilité : elle permet d'injecter des fautes à divers niveaux d'abstraction et est totalement indépendante du simulateur utilisé. Néanmoins, cette approche a un coût : la modification du code source pour chaque faute peut être laborieuse, et cela s'ajoute au temps nécessaire pour la simulation.

Un saboteur, en revanche, est un composant ajouté au RTL avec le seul objectif d'injecter des fautes dans le système. En général, un saboteur reste inactif jusqu'à ce qu'un ou plusieurs signaux spécifiques soient activés, déclenchant ainsi la modification du circuit au niveau des signaux ou des timings. Ces saboteurs peuvent être intégrés manuellement ou automatiquement dans le RTL, et peuvent être placés en parallèle ou en série avec les composants existants. Selon des travaux antérieurs [Nav+11], les saboteurs permettent de simuler un large éventail de fautes et de conditions environnementales, comme le bruit électromagnétique. Cependant, leur capacité à modéliser des fautes est limitée à celles qui affectent les signaux et les timings, et non celles qui se produisent au niveau des portes logiques.

La seconde famille d'approches repose sur l'injection de fautes dans les traces de simulation. Cette méthode peut être mise en œuvre à l'aide d'un simulateur SPLICE pour les niveaux de transistors ou à travers des simulations au niveau RTL. Deux techniques sont généralement employées dans ce contexte. La première consiste à modifier les signaux : on déconnecte le signal de son driver et on le force à adopter une valeur fixe. La seconde technique implique la modification des variables au sein d'un processus en cours d'exécution. Il est important de noter que l'efficacité de l'injection de fautes par les outils de simulation dépend fortement des caractéristiques spécifiques de ces outils.

Les techniques de simulation peuvent être évaluées selon trois critères principaux :

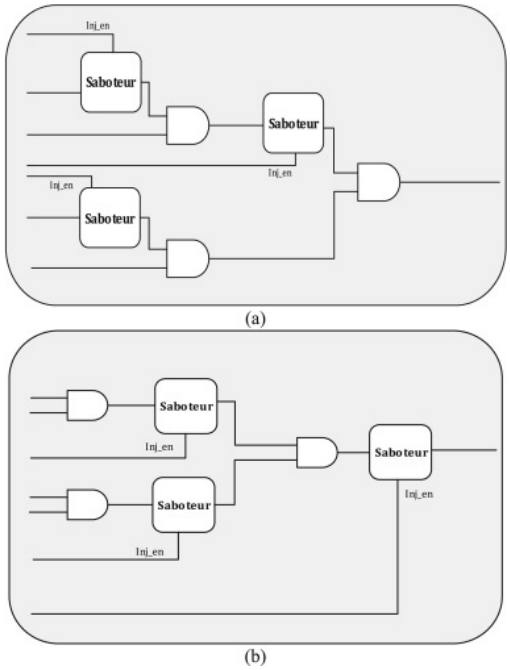

Figure 2.5 : Placement des saboteurs en entrée ou en sortie.

- La capacité à simuler des fautes : ce critère évalue la diversité et la précision des types de fautes que la technique peut simuler.
- L'effort de mise en place : ce critère mesure la complexité et le temps requis pour implémenter la technique dans un environnement de simulation donné.
- L'overhead de temps de simulation : ce critère évalue l'impact de la technique sur la durée totale de la simulation, notamment en termes de temps supplémentaire nécessaire pour injecter les fautes.

Ces critères fournissent un cadre pour l'analyse comparative des différentes techniques de simulation, facilitant ainsi le choix de la méthode la plus appropriée pour un contexte spécifique.

- En termes de capacité à simuler des fautes, les mutants se distinguent par leur flexibilité et leur précision, offrant la meilleure performance dans ce domaine.
- Concernant l'effort de mise en place, la manipulation des signaux et des variables est la plus simple à implémenter. En revanche, les mutants et les saboteurs exigent un investissement plus conséquent, notamment en ce qui concerne la création et la génération de nouveaux modèles, ainsi que la recompilation du RTL. Toutefois, les

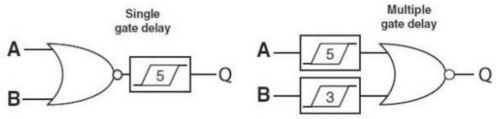

Figure 2.6 : Ajout des informations de delays en entrée ou en sortie de porte.

saboteurs sont généralement plus simples à mettre en œuvre que les mutants.

- Pour ce qui est de l'overhead de temps de simulation, la manipulation des variables et des signaux peut nécessiter des arrêts et des redémarrages de la simulation, ce qui augmente le temps total de l'exercice.

D'autre part, l'overhead de temps de simulation pour les mutants et les saboteurs dépend de plusieurs facteurs : la quantité d'événements additionnels introduits, le nombre de lignes de code exécutées pour chaque événement, et la complexité du contrôle ajouté au système. Ces éléments doivent être pris en compte lors de l'évaluation de la méthode la plus adaptée à un projet donné.

Injection par émulation

L'injection par simulation sur FPGA se concentre sur les méthodes spécifiquement adaptées à cette technologie, excluant ainsi les approches de mutants et de saboteurs. Un défi majeur dans la vérification de la résilience aux fautes sur FPGA, surtout lorsque la cible finale est un ASIC, réside dans les différences de délais entre ces deux technologies. Ces différences rendent difficile la généralisation des résultats obtenus sur FPGA aux cibles ASIC.

Pour pallier ce problème, des informations supplémentaires relatives au temps et à la consommation sont souvent intégrées dans le modèle de simulation. Ces données permettent de mieux appréhender et de minimiser les effets de masquage qui peuvent survenir lors de l'injection de fautes dans la logique du circuit. Des travaux antérieurs ont abordé cette question en ajoutant des informations temporelles [Val+07] et des données sur la consommation d'énergie [Ent+09], afin de rendre les simulations sur FPGA plus représentatives des comportements que l'on observerait sur un ASIC, comme illustré dans la figure 2.6.

L'utilisation de la reconfiguration dynamique des FPGA comme outil d'injection de faute est une approche qui a gagné en popularité depuis son introduction en 2000 [ALF02]. Elle est aujourd'hui considérée comme l'un des domaines les plus actifs dans la simulation

des fautes [Fib+19]. L'un des principaux avantages de cette technique réside dans son utilisation des mécanismes internes de reconfiguration du FPGA pour injecter des fautes. De plus, la reconfiguration peut être partielle, ce qui permet d'injecter une faute à une position spécifique tout en laissant le reste du circuit inchangé. Le temps nécessaire pour ce processus de reconfiguration est également nettement inférieur à celui requis pour une synthèse complète.

Toutefois, cette méthode présente également des inconvénients. Étant donné qu'elle s'applique à une cible matérielle, il peut être difficile de tracer la chaîne de conséquences résultant de l'injection de la faute. En d'autres termes, ce que l'on gagne en rapidité de simulation, on le perd en visibilité et en compréhension du comportement du système en présence de fautes.

2.2.4 Modèle de faute

Le modèle de faute est essentiel pour comprendre et évaluer l'impact des perturbations sur un circuit électronique. Il offre une représentation abstraite qui décrit comment une faute affecte le comportement global d'un système. Dans le contexte de cette thèse, nous adoptons une classification inspirée de celle proposée par [Ott05]. Bien que d'autres classifications existent, celle-ci a été choisie pour sa clarté et sa couverture exhaustive des différents modèles de fautes. Toutefois, il est important de noter que nous avons apporté quelques modifications à cette classification pour y inclure des détails supplémentaires concernant les fautes multiples. De plus, les définitions des fautes transitoires et permanentes ont été ajustées pour correspondre à celles présentées en 2.2.1.

Pour caractériser une faute, plusieurs paramètres sont pertinents :

- Localité spatiale : Les fautes peuvent être aléatoirement (répartie aléatoirement sur le circuit), localisées (une variable peut être ciblée précisément) et précisent (un octet ou un bit peut être ciblé)
- Localité temporelle : Les fautes peuvent être aléatoirement (répartie aléatoirement sur le circuit), localisées (une variable peut être ciblée précisément) et précisent (un octet ou un bit peut être ciblé).
- Nombre de bits affecté
- Durée : temps pendant laquelle la faute a lieu si cette faute est transitoire ou bien permanente.

Paramètres	Valeurs possibles
Localité spatiale	
Localité temporelle	
Fautes multiples	
Nombre de bits affecté	
Durée	
Type de fautes	
Probabilité	

Table 2.3 : tableau type d'un modèle de faute.

- Type de fautes : plusieurs types de fautes peuvent intervenir selon le circuit et la méthode d'injections de fautes. Chacune de ces fautes peut être transitoire ou permanente.
  - Stuck-at fault : la valeur des bits affectés ne changera plus.
  - Bit flip : la valeur des bits affectés prend la valeur complémentaire à l'instant de la faute.
  - Random fault : la valeur des bits affectés prend une valeur aléatoire.
  - Bit set : la valeur des bits affectés prend une valeur 1.
  - Bit reset : la valeur des bits affectés prends une valeur 0.
- La probabilité : avec quelle probabilité une faute peut avoir lieu. Par exemple certaines attaques ont plus de chance de produire des "bit set" que des "bit reset" sur les bits notamment les attaques par glitch.
- Localité spatiale des fautes multiples : cette possibilité est ajoutée pour introduire des fautes multiples induites par un évènement commun. Cet ajout caractérise le nombre de bits consécutifs affectés lors d'une faute. En effet, lorsqu'une faute a lieu il y a des chances que les transistors proches subissent aussi cette faute.

Pour résumer les principales caractéristiques, des paramètres sont donnés dans le tableau 2.3.

Deux types grandes types de fautes se distinguent celui des soft errors et celui des injections de fautes. La principale différence entre les deux est qu'il n'y a pas de soft errors multiples du fait de la nature aléatoire de celle ci. En effet, il est improbable que deux particules chargées tombent en même temps sur la même variable.

Localité spatiale	Aléatoire
Localité temporelle	Cycle
Fautes multiples	Non
Nombre de bits affecté	Entre 1 - longueur de la variable affecté
Durée	99.99% Trans 0.01% Perm
Type de fautes	Bit-flip
Probabilité	100%

Table 2.4 : tableau type d'un modèle dans un contexte de sureté.

Localité spatiale	Sur les variables sensibles
Localité temporelle	Cycle
Fautes multiples	Jusqu'à 4
Nombre de bits affecté	Entre 1 - longueur de la variable affecté
Durée	transitoire
Type de fautes	Bit-flip
Probabilité	100%

Table 2.5 : tableau type d'un modèle dans un contexte de sécurité.

Sureté : Soft error

Dans le contexte de la sûreté, l'objectif est de corriger les fautes le plus rapidement possible. Ces fautes sont considérées comme des événements normaux dans le fonctionnement du circuit et surviennent de manière aléatoire. Avec la miniaturisation des circuits électroniques, l'impact des rayonnements cosmiques, entraînant des fautes multi-bits, est devenu de plus en plus probable [ZES89] [Pag+11]. Ces fautes multiples sont spatialement proches, mais leur localisation spatiale est aléatoirement distribuée sur l'ensemble du circuit. Les caractéristiques de ces types de fautes sont présentées dans le tableau 2.4.

Sécurité : Injection de faute

Dans le contexte de la sécurité, l'objectif est d'évaluer la résistance du système contre un attaquant doté de capacités à l'état de l'art. Un tel attaquant dispose de moyens d'injection de fautes avancés, notamment la capacité d'effectuer des injections laser multiples (jusqu'à 4) avec un contrôle précis de la puissance de chaque laser. Cela lui permet d'être précis au niveau du cycle d'horloge et de la variable ciblée. Grâce à la variation de la puissance du laser, il peut également affecter plusieurs bits consécutifs. Les caractéristiques de ce type d'attaquant sont résumées dans le tableau 2.5.

2.2.5 Solutions

Les contremesures envisagées doivent être totalement transparentes pour le logiciel. Bien qu'il soit possible d'introduire de nouvelles instructions RISCV pour ajuster les niveaux de sécurité, celles-ci ne devraient servir qu'à activer ou désactiver explicitement des composants matériels. Notre focus se porte uniquement sur les contremesures au niveau microarchitectural, car elles sont aisément adaptables à divers designs, indépendamment de la technologie d'implémentation. Les méthodes de détection basées sur des capteurs et les solutions au niveau des portes logiques ne sont pas considérées ici. Néanmoins, elles peuvent compléter efficacement les travaux microarchitecturaux.

Les stratégies majeures pour renforcer un système face aux attaques par injection de fautes reposent sur la redondance. Cette dernière peut être classifiée en trois catégories : spatiale, temporelle et informationnelle. Un élément crucial à considérer est le niveau de redondance. Une redondance simple permet uniquement de détecter les fautes. En revanche, des niveaux de redondance plus élevés offrent la capacité de détecter et de corriger ces fautes. Comme mentionné précédemment, classifier le potentiel de détection et de correction d'une solution s'avère complexe. Ainsi, bien que le niveau de redondance soit pertinent pour les redondances spatiales et temporelles, son application à la redondance informationnelle est moins directe.

Redondance par réplication

Les méthodes traditionnelles de sécurisation contre les fautes reposent sur la duplication des blocs fonctionnels et la comparaison de leurs résultats. Cette approche peut être appliquée à divers niveaux, allant d'une simple porte logique dans un étage de pipeline jusqu'au processeur dans son intégralité.

La duplication matérielle implique une copie à l'identique des composants concernés. En associant cette duplication à un comparateur, il est possible de détecter des erreurs. Cette méthode est efficace pour repérer des erreurs survenant sur une seule des unités dupliquées. Cependant, si les deux unités sont fautives, certaines erreurs peuvent échapper à la détection. Pour aller au-delà de la simple détection et permettre la correction des erreurs, un triplement des unités fonctionnelles est nécessaire, accompagné d'un mécanisme de vote pour évaluer la validité des données.

Toutefois, cette approche présente des limites. Bien qu'elle soit efficace contre les fautes aléatoires, elle est vulnérable aux attaques ciblées. En effet, un attaquant pourrait théori-

quement induire des fautes simultanées sur deux des unités redondantes, rendant la faute indétectable.

Il est également important de noter que les mécanismes matériels ajoutés pour renforcer la sécurité sont eux-mêmes susceptibles de fautes. Une solution simple à ce problème consiste à dupliquer également les mécanismes de vote. Ces stratégies sont communément appelées Full TMR (Triple Modular Redundancy), car elles impliquent le triplement non seulement du composant en question, mais aussi du mécanisme de vérification.

Ces méthodes de redondance sont particulièrement populaires dans le domaine de la fiabilité, où les fautes sont généralement distribuées de manière aléatoire. Ainsi, la probabilité que deux fautes affectent simultanément deux unités redondantes est extrêmement faible. Cependant, dans le contexte de la sécurité, un attaquant pourrait délibérément injecter des fautes identiques sur les unités redondantes, compromettant ainsi le système.

La duplication temporelle est une autre technique de sécurisation qui consiste à exécuter la même opération à plusieurs reprises et à comparer les résultats obtenus. Tout comme pour la duplication matérielle, une duplication temporelle simple permet uniquement la détection des erreurs, tandis qu'un triplement permet à la fois la détection et la correction.

Cependant, cette méthode présente des vulnérabilités, notamment dans le contexte des processeurs cryptographiques. Une faute de durée suffisamment longue peut affecter les deux itérations successives de la même manière, générant ainsi une faute considérée comme valide. Ce type de scénario met en évidence les limites de la duplication temporelle face à des attaques ciblées et sophistiquées.

Redondance d'information

La redondance informationnelle constitue une autre approche de sécurisation. Elle implique l'ajout d'informations supplémentaires aux données afin de permettre la détection ou la correction des fautes. Les principales techniques en la matière sont les codes détecteurs et correcteurs d'erreurs. Toutefois, peu de ces codes sont adaptés au contexte matériel, et encore moins aux contraintes spécifiques des pipelines. Les codes les plus couramment utilisés dans ce contexte sont les codes résidus, le code Berger et la parité.

L'un des avantages de ces codes est qu'ils ne reposent pas sur une simple duplication des données, rendant ainsi plus difficile pour un attaquant de cibler une zone spécifique pour induire une faute. De plus, certains codes correcteurs rendent impossible l'injection d'er-

reurs mono-bit, car ces erreurs seront inévitablement détectées. Un autre avantage majeur des codes correcteurs est leur légèreté relative par rapport à la duplication, ainsi que leur moindre susceptibilité aux fuites d'informations par des canaux auxiliaires.

Parmi les codes détecteurs et correcteurs d'erreurs les plus utilisés dans le domaine matériel, on peut citer la parité, le Cyclic Redundancy Check (CRC), le code de Hamming, le code BCH, le code SEC-DED, le code Reed-Solomon, Reed-Muller et les Low-Density Parity Codes (LDPC).

Offuscation

Enfin, l'obfuscation représente une autre grande catégorie de protection contre les injections de fautes. Dans le contexte des cryptoprocesseurs, cette technique peut inclure l'ajout de cycles factices, durant lesquels le processeur exécute des opérations non pertinentes pour le calcul en cours. Une autre stratégie consiste à mélanger les données pour rendre plus difficile pour l'attaquant de déterminer où injecter des fautes.

Il est important de noter que ces méthodes d'obfuscation doivent impérativement être combinées avec d'autres techniques de protection pour être efficaces. De plus, leur efficacité dépend de leur caractère aléatoire. Si l'obfuscation est basée sur une constante, l'attaquant n'aura qu'à identifier cette constante pour contourner la protection. En revanche, si l'obfuscation est aléatoire, l'attaquant devra répéter son processus d'identification pour chaque nouvelle attaque.

En somme, bien que ces méthodes d'obfuscation soient utiles pour augmenter la complexité de l'injection de fautes, elles ne constituent pas en elles-mêmes une protection complète contre les attaques.

2.3 Conclusion

Ce chapitre a offert une vue d'ensemble des attaques par observation et par perturbation, ainsi que des concepts généraux relatifs aux contre-mesures.

Les attaques par canaux auxiliaires sont complexes et diversifiées, avec de nombreux vecteurs de fuite et des méthodes variées pour exploiter ces fuites. Un corpus théorique substantiel a été développé pour évaluer les techniques de masquage, et diverses méthodes empiriques sont proposées pour évaluer d'autres types de contre-mesures. Dans le cadre

de cette thèse, nous nous focaliserons sur les contre-mesures basées sur la randomisation et l'ajout de bruit.

Les attaques par injection de fautes sont également diverses, mais elles souffrent d'un manque de méthodes formelles et de métriques pour évaluer l'efficacité des contre-mesures. De plus, il est complexe de garantir simultanément la sécurité et la sûreté d'un composant. Bien que cette thèse se concentre principalement sur la sécurité, des pistes de réflexion et d'implémentation seront proposées pour intégrer à la fois la sûreté et la sécurité dans le contexte des fautes sur les circuits électroniques.

Enfin, une analyse détaillée des contre-mesures possibles révèle qu'elles sont en partie antinomiques. Les techniques de masquage pour les canaux auxiliaires augmentent les opportunités d'injecter des fautes, tandis que les redondances introduites pour se prémunir contre les fautes créent de nouvelles fuites d'information. Ce constat souligne la complexité inhérente à la conception de systèmes à la fois sûrs et sécurisés.

3

Chemin de donnée

Ce chapitre aborde les vulnérabilités et les risques liés au chemin de données du pipeline dans les architectures de processeurs généralistes. Notre principale contribution concerne la préservation de l'intégrité des données tout au long de leur passage dans le pipeline. À cette fin, nous introduisons un mécanisme de tag d'intégrité qui permet non seulement de vérifier l'intégrité des données elles-mêmes, mais également celle des calculs effectués lors de l'étape d'exécution (EXEC).

3.1 Modèle d'attaquant

Les architectures de processeurs généralistes, fréquemment utilisées dans les systèmes embarqués et les serveurs, nécessitent des mécanismes de sécurité robustes pour garantir l'intégrité des données et des calculs. Avec l'essor de l'Internet des Objets (IoT) et des systèmes cyber-physiques, les attaques par injection de fautes gagnent en popularité, notamment en raison de la vulnérabilité intrinsèque du chemin de données. Ce dernier est en effet le cœur du traitement des données et des instructions, et est donc une cible privilégiée pour les attaques par injection de fautes [Men+20]. Ces attaques peuvent induire des erreurs dans les registres internes du pipeline, affectant ainsi soit le flux d'exécution, soit les résultats des opérations arithmétiques et logiques.

Les attaques par canaux auxiliaires présentent une menace significative pour le chemin de données des processeurs généralistes. Les émissions EM, inhérentes à l'activité électrique des composants, peuvent refléter le traitement des données en cours. Par exemple, Longo et al. [Lon+15] ont démontré que les émissions EM peuvent être analysées pour extraire des clés cryptographiques lors des opérations de chiffrement sur des processeurs.

On considère que notre attaquant possède des capacités d'injections de fautes Cet attaquant possède des aptitudes élevées en matière d'injection de fautes, notamment via l'utilisation d'un dispositif laser multi-spots, générant des fautes multiples non stochastiques et additives (en appliquant une fonction xor sur le bit altéré). Bien que nous ne fixions pas de limite sur le nombre de fautes, en pratique. Le scénario des erreurs stochastiques n'est pas exploré, puisque notre solution affiche des performances comparables à celles d'une duplication dans ce contexte. De plus, la capacité d'un laser à engendrer un grand nombre de fautes, en augmentant la puissance du laser utilisé, confère à l'attaquant une vaste gamme de fautes potentielles.

Dans le cas plus spécifique d'un processeur à usage général, nous supposons que l'attaquant cible uniquement le chemin des données, c'est-à-dire là où seules les données circulent, de sorte que les chemins d'instruction et de contrôle ne sont pas dans le champ d'action de notre attaquant. Nous traiterons ce cas dans le chapitre suivant. Cependant, le chargement et le stockage des données, les opérations logiques et arithmétiques, ainsi que le fichier de registres et les registres inter-étages sont tous des cibles potentielles. En plus des attaques par injection de fautes, notre attaquant dispose de capacités d'attaque par canal auxiliaire. Nous nous limitons au premier ordre d-probing, car les contre-mesures d'ordre supérieur sont prohibitivement coûteuses pour le pipeline du processeur. Le modèle d'ad-

versaire inclut également les défaillances matérielles. Par conséquent, il est supposé que les données ont pu traverser la hiérarchie mémoire avec l'assurance de la confidentialité, intégrité et authenticité des données et arriver sous forme masquée dans les registres généraux du CPU. La possibilité d'un chiffrement DRAM authentifié [SEH20] mais aussi de protéger la hiérarchie de cache avec un schéma de masquage léger [Tal+22] a déjà été démontrée. Nous supposerons donc que les données sont masquées dans les registres généraux du CPU, avec leurs étiquettes d'authenticité (qui sont elles-mêmes masquées) et les masques associés.

3.2 Contre-mesures existantes

Les contre-mesures contre les attaques par canal auxiliaire dans les architectures de processeurs généralistes mettent principalement l'accent sur des techniques de masquage de données. Ces dernières sont alors distribuées entre plusieurs "shares" aléatoires [PR13] dans le but de rendre la fuite d'information indépendante de la donnée sensible. Chaque "share" est ainsi traité tout en garantissant une indépendance stochastique avec les autres. Désormais, diverses techniques de masquage sont envisageables et offrent une résistance aux glitches, telles que TI ou DOM. Dans cette thèse [Gro18], l'auteur applique le masquage DOM à un processeur RISCV.

Pour ce qui est des contre-mesures contre les injections de fautes, la majorité des solutions disponibles sont spécifiquement conçues pour des environnements hautement spécialisés, comme les processeurs durcis pour les applications spatiales. Par exemple, [Bar+21] fusionne différentes solutions simples, en particulier la combinaison de la redondance spatiale et temporelle. En effet, avec leur architecture multihart pipeline. Le terme "Hart" (Hardware Thread) représente un thread d'exécution matériel, qui comprend son propre ensemble de registres et un compteur de programme, mais qui peut partager d'autres ressources système, comme la mémoire et les unités de traitement, avec d'autres Harts au sein du même processeur. Dans ce processeur un hart différent est exécuté à chaque fois, évitant ainsi les cycles de gel et rendant tous les étages indépendants. Ils peuvent donc intégrer dans un même étage deux cycles différents sans problème de dépendance. En disposant d'une Dual Modular Redundancy une duplication des unités de traitement, il est possible d'obtenir une Triple Modular Redundancy de deux instructions sur trois cycles. Ainsi, un compromis est établi entre le surcoût en surface et en vitesse d'exécution. L'avantage notable de cette méthode réside dans la possibilité de bénéficier de l'aspect activable/désac-

tivable sans surcoût de la redondance temporelle.

Dans le contexte des processeurs généralistes, il existe un manque notable de solutions pour contrer efficacement les injections de fautes. Ce déficit en matière de sécurité souligne le besoin urgent de développer des contre-mesures plus complètes et économiquement viables. En résumé, bien que des solutions existent pour chacune de ces catégories d'attaques, il demeure un besoin pressant de méthodes de protection plus adaptées aux défis uniques posés par les architectures de processeurs généralistes modernes.

3.3 Tag homomorphique

3.3.1 Présentation de la contre-mesure

Dans ce chapitre, nous proposons de scinder l'étage d'exécution du CPU en deux domaines parallèles, comme illustré sur la Figure 3.1. Le premier domaine, qualifié de canonique, assure l'exécution des opérations logiques et arithmétiques de l'ALU en routine. Parallèlement, le second domaine, nommé domaine "permuté", exécute les mêmes opérations sur des données ayant subi une permutation bit à bit. Ce domaine agit en tant que tag d'authenticité ou MAC, la permutation étant dictée par une clé secrète. La clé a été sélectionnée pour sa structure arborescente, la dotant de propriétés homomorphes vis-à-vis des opérations logiques et arithmétiques, facilitant ainsi des vérifications de calculs très rapides. En dépit de la complexité d'implémentation d'une permutation en hardware, nous démontrons que son encombrement spatial est minime et son utilisation hautement efficiente pour la majorité des opérations. L'association de cette permutation à une clé secrète interne au CPU, modifiable à souhait, confère une forme de polymorphisme ou de randomisation à l'implémentation matérielle de l'ALU. Ceci entrave considérablement l'injection de fautes ciblées par un attaquant.

Les qualités homomorphes de cette permutation la rendent également compatible avec le masquage à tous les ordres, bien que nous envisagions, dans la suite, un masquage au premier ordre (voir Figure 3.2). L'efficacité du schéma employé implique qu'il ne peut prétendre à une qualité cryptographique élevée ; en effet, la connaissance de quelques couples entrée/sortie de la permutation peut suffire à révéler la clé. Toutefois, son usage interne au CPU, associé à un masquage et à une modification fréquente de la clé, offre une assurance CIA (Confidentialité, Intégrité, Authenticité) tout à fait satisfaisante.

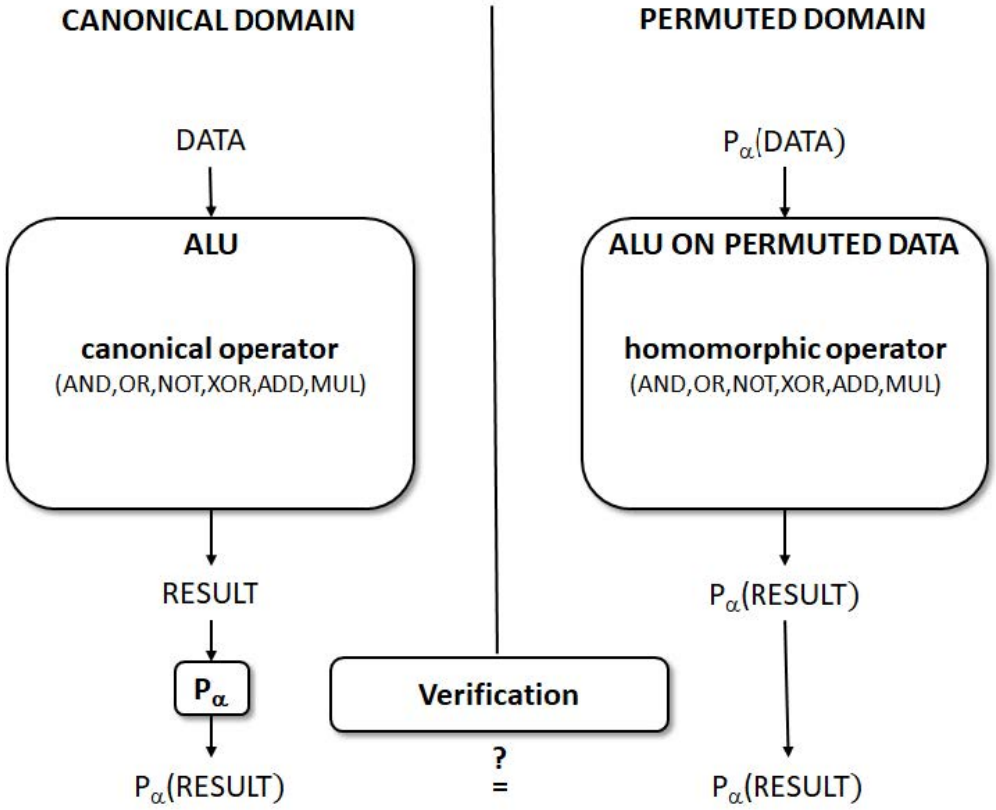

FIGURE 3.1 : Schéma de chemin de données. Les calculs dans l'ALU sont divisés en deux domaines masqués : un domaine canonique où les opérations de l'ALU sont effectuées comme d'habitude et un domaine permuté où les données ont subi une permutation bit à bit qui devient un tag d'authenticité. Dans ce domaine permuté, des opérateurs homomorphiques sont utilisés dans une ALU spécifique qui travaille sur les tags.

3.3.2 Choix de la permutation

Pour démontrer l'exactitude des calculs effectués au sein du CPU, des calculs parallèles peuvent être réalisés sur leurs tags d'authenticité. En se basant sur l'axiome selon lequel toute fonction de logique combinatoire peut uniquement être exécutée avec des portes NAND (la porte NOR partage cette propriété), nous avons exploré des fonctions manifestant un homomorphisme relativement à cette porte logique. Pour rappel, une fonction est dite homomorphe par rapport à une opération donnée op s'il existe une opération op' telle que pour tous x et y :

$$f(x\, op\, y) = f(x)\, op'\, f(y)$$

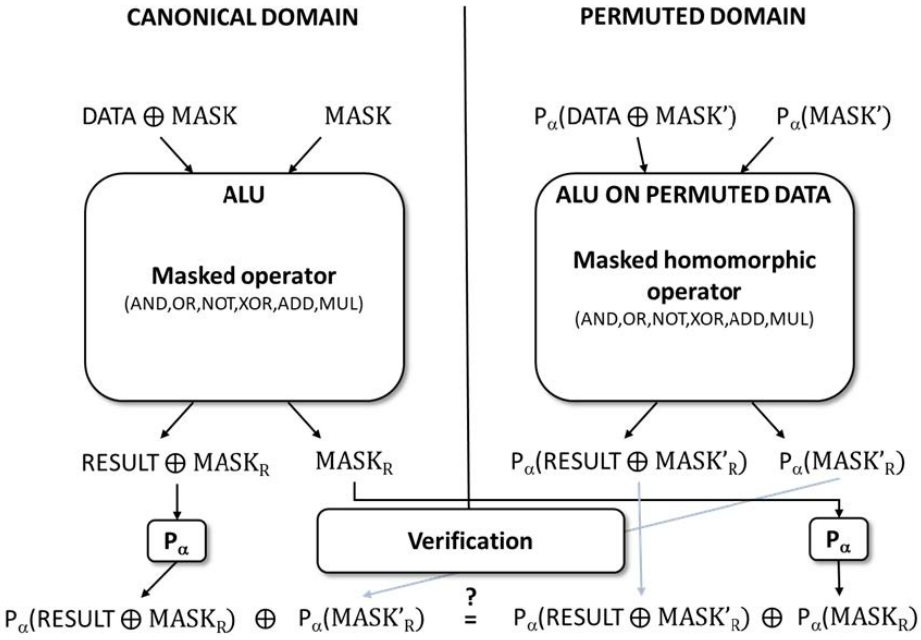

Figure 3.2 : Schéma de chemin de données. Les deux domaines peuvent être masqués pour apporter de la confidentialité. La fonctionnalité homomorphique de la permutation simplifie toujours la vérification des calculs.

Concernant l'opération logique NAND, on peut plus simplement chercher $op = op' = NAND$. On constate alors que la vérification des calculs entre $x\,op\,y$ et $f(x)\,op'\,f(y)$ devient favorable car il suffit d'appliquer f sur $x\,op\,y$ et de vérifier l'équivalence avec $f(x)\,op'\,f(y)$. Uniquement deux classes de fonctions mathématiques permettent d'accomplir cet homomorphisme sur des données multi-bits : il s'agit soit des projections p définies par $p^2 = p$ soit des permutations bit à bit définies par $p^2 = \mathrm{Id}$. Les projections sont dépourvues d'intérêt pour générer des tags d'intégrité puisqu'elles entraînent intrinsèquement une perte d'information sur les données.

La permutation, malgré sa complexité intrinsèque d'implémentation en logique binaire, se présente-t-elle comme une candidate viable pour la création d'un tag d'authenticité ? Il est notable que toute tentative d'attaque par injection de fautes est identifiable par une permutation. Néanmoins, il est a priori impraticable de déterminer la permutation adéquate à utiliser avant l'occurrence de l'attaque. Par ailleurs, avec l'avènement des architectures open source, la probabilité que l'attaquant détienne une connaissance approfondie du système s'accroît, lui permettant ainsi d'élaborer des scénarios d'injections de fautes qui pourraient contourner les redondances. Même en possédant une compréhension exhaus-

tive de l'architecture, l'attaquant demeure incapable de prédire l'aléatoire. L'intégration d'aléa complexifie donc l'attaque d'un facteur irréductible du point de vue de l'attaquant. Ce facteur irréductible nous autorise à instaurer une notion de résilience et de réaction face à la menace dans la durée impartie.

La permutation mise en œuvre a donc été élaborée pour être dépendante d'une clé secrète tout en maintenant des propriétés attrayantes pour être également homomorphe relativement aux opérations arithmétiques, et non uniquement logiques. Des permutations entre 2 bits uniquement (que nous désignerons sous le terme de transposition) ou entre deux blocs de 2^k bits ont été sélectionnées. Celles-ci peuvent être effectuées au moyen de la porte de Fredkin, fréquemment employée en reversible computing et en quantum computing, qui, au triplet (a, b, c), associe le couple $(a.\bar{c} + b.c, a.c + b.\bar{c})$ où $\bar{c}$ est le complémentaire de c. En d'autres termes, cette porte renvoie (a, b) vers (a, b) si le bit de clé c est égal à 0, et vers la permutation bit à bit (b, a) si le bit de clé c est égal à 1. Le bit c endosse ainsi la fonction de la clé de la permutation.

Cette porte manifeste la particularité de révéler minimalement la clé c lors d'attaques side-channel, en raison d'une dépendance symétrique de la porte vis-à-vis de c et $\bar{c}$. Toutefois, ce schéma de "chiffrement" ne s'avère pas très robuste, car, en possédant quelques valeurs de texte chiffré, il devient aisément possible de discerner comment les bits sont permutés, et donc de remonter à la clé secrète. Cette technique est ainsi réservée au fonctionnement interne du CPU et à la hiérarchie de caches, et ne devra pas être employée dans une mémoire externe dont les données pourraient être extraites aisément. Ainsi, tout comme pour le M&M, il est envisageable de masquer les données et la donnée permutée. De ce fait, au lieu d'avoir $(x, p_\alpha(x))$ où α est la clé secrète de la permutation p_α, on opère avec $(x \,\text{XOR}\, m, m, p_\alpha(x) \,\text{XOR}\, m', m')$ pour contrer efficacement les fuites side-channel. Il est également à noter que p_α étant homomorphe au XOR, on a $p_\alpha(x) \,\text{XOR}\, m' = p_\alpha(x \,\text{XOR}\, m'')$ où $m' = p_\alpha(m'')$.

3.3.3 Permutation par bloc

Il existe de nombreuses méthodes pour effectuer des permutations dépendantes d'une clé. Toutefois, certaines se prêtent davantage à faciliter les opérations complexes telles que les décalages, additions et multiplications, au détriment de quelques concessions au niveau de l'entropie de la permutation. La solution adoptée a été celle d'un arbre à structure dichotomique. Cet arbre repose sur une division en blocs de notre donnée ; par exemple, pour une

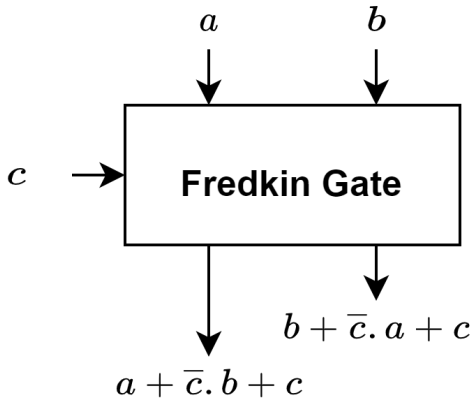

Figure 3.3 : Porte de Fredkin.

donnée de 32 bits, on peut la diviser en 2 blocs de 16 bits, ces deux blocs de 16 bits peuvent à leur tour être divisés en 4 blocs de 8 bits. Il est possible de poursuivre ainsi jusqu'à obtenir des blocs de 1 bit. Avec cette division en blocs, on peut employer la porte de Fredkin pour réaliser une transposition de deux blocs de même taille selon une clé, comme illustré en figure 3.4. Une transposition T_K^n permute le contenu de blocs de taille n lorsque $K = 1$. Seuls

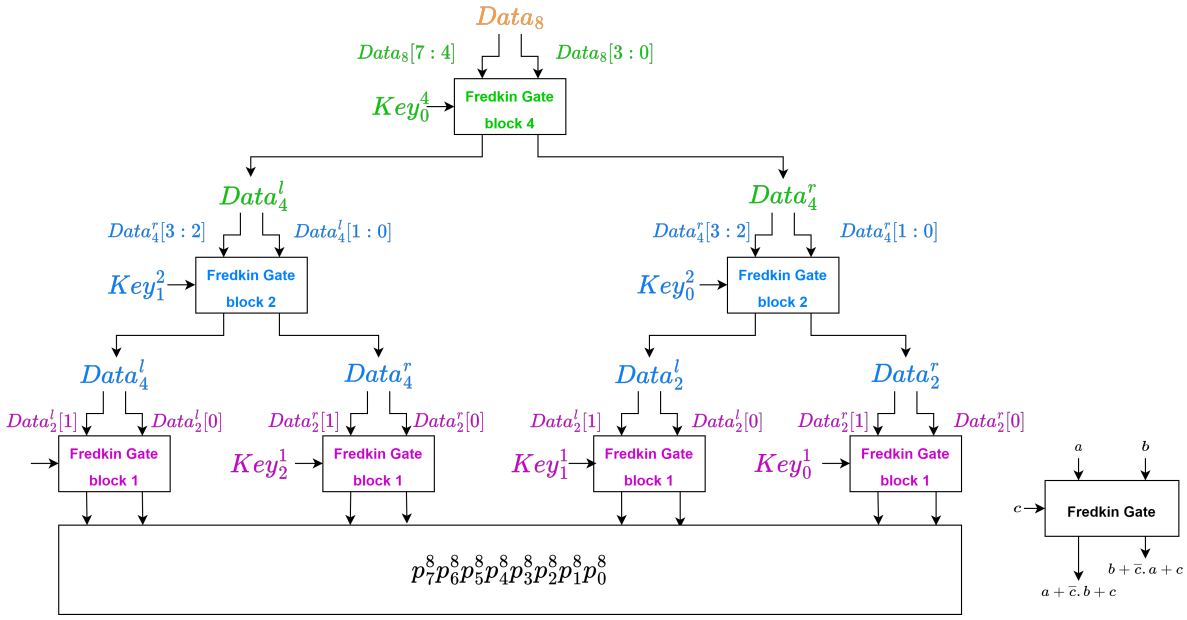

Figure 3.4 : Arbre de permutation par bloc.

deux blocs consécutifs sans chevauchement sont transposés. Par exemple, pour un mot de 32 bits, 1 bit de clé est requis pour transposer les 2 blocs de 16 bits, 2 bits pour les 4 blocs de 8 bits, 4 bits pour les 8 blocs de 4 bits, etc. La clé est ainsi la concaténation des différentes

clés de permutation $\text{Key} = \text{Key}_0^{16}|\text{Key}_1^8|\text{Key}_0^8|\text{Key}_3^4|\ldots|\text{Key}_0^4|\text{Key}_7^2|\ldots|\text{Key}_0^2|\text{Key}_{15}^1|\ldots|\text{Key}_0^1$ ayant une longueur de 31 bits pour une permutation de 32 bits P_{Key}^{32}. Elle est structurée en 5 niveaux de transpositions, totalisant 31 transpositions agencées en débutant par les blocs les plus larges $P_{\text{Key}}^{32} = T_{\text{Key}_0^1}^1 \circ \ldots \circ T_{\text{Key}_{15}^1}^1 \circ T_{\text{Key}_0^2}^2 \circ \ldots \circ T_{\text{Key}_7^2}^2 \circ T_{\text{Key}_0^4}^4 \circ \ldots \circ T_{\text{Key}_3^4}^4 \circ T_{\text{Key}_0^8}^8 \circ T_{\text{Key}_1^8}^8 \circ T_{\text{Key}_0^{16}}^{16}$ comme illustré en Figure 3.5. Chaque sortie de transposition du niveau n sert d'entrée pour la transposition suivante de niveau $n/2$. Les transpositions d'un niveau donné peuvent être réalisées dans n'importe quel ordre, étant donné que les chevauchements à un niveau n'ont pas été autorisés. L'avantage et la lacune de cette permutation résident dans sa conservation de la localité par blocs. Les n bits formant un bloc demeurent invariablement dans le même bloc en sortie de permutation. Cette caractéristique accorde davantage d'informations à l'attaquant, tout en facilitant l'exécution des opérations complexes telles que les décalages et les additions.

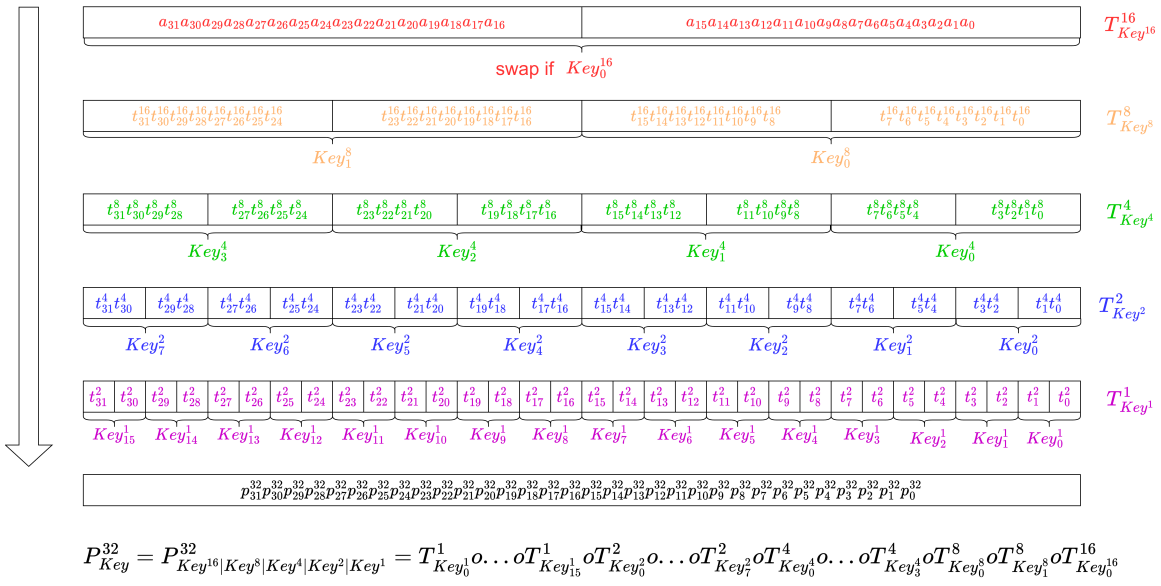

FIGURE 3.5 : Permutation par bloc.

3.4 Opération logiques et arithmétiques

3.4.1 Opérations booléennes

Dans un processeur généraliste, l'Unité Arithmétique et Logique (ALU) exécute des opérations booléennes, logiques et arithmétiques. Pour renforcer un processeur contre les fautes, il est essentiel que ces opérations puissent être menées sur des opérandes qui ont été permutés comme établi dans 3.3.3. La protection des opérations arithmétiques d'un processeur implique la réalisation de ces opérations dans le domaine de la redondance. La principale difficulté en l'occurrence réside dans le fait que cette représentation se traduit par une permutation des bits selon une clé. Le choix de la permutation, structurée en transpositions de blocs, garantit que toutes les opérations booléennes bit à bit telles que AND, OR, NOT, NAND, NOR et XOR puissent être effectuées simultanément sur les données et sur les tags d'authenticité sans modification d'opérateur. Il convient donc d'aborder les trois opérations les plus courantes : les décalages, l'addition et la multiplication.

3.4.2 Addition

La première opération à réaliser est l'addition, dont la principale complexité réside dans la propagation de la retenue. Effectivement, la retenue se propage de bit en bit, et il est observable que, de par la division en blocs de 2^n bits, cette permutation est particulièrement bien adaptée aux structures en arbre dichotomique, tout comme le décalage décrit précédemment. Il existe déjà des structures permettant une propagation dichotomique rapide de la retenue, comme c'est le cas avec le Carry Look-Ahead.

Carry look ahead L'essence de cet additionneur repose sur l'anticipation de la retenue à chaque étage avant le calcul de la somme. L'obtention d'une retenue sortante à un étage d'addition est tributaire de la génération et la propagation de la retenue. Ces aspects sont manifestes dans l'expression de la retenue d'indice i, définie comme suit :

$$c_{i+1} = (a_i \cdot b_i) + (a_i + b_i) \cdot c_i$$

.

Ainsi, nous avons :

- $G_i = (a_i \cdot b_i)$ désigné comme le générateur de la retenue

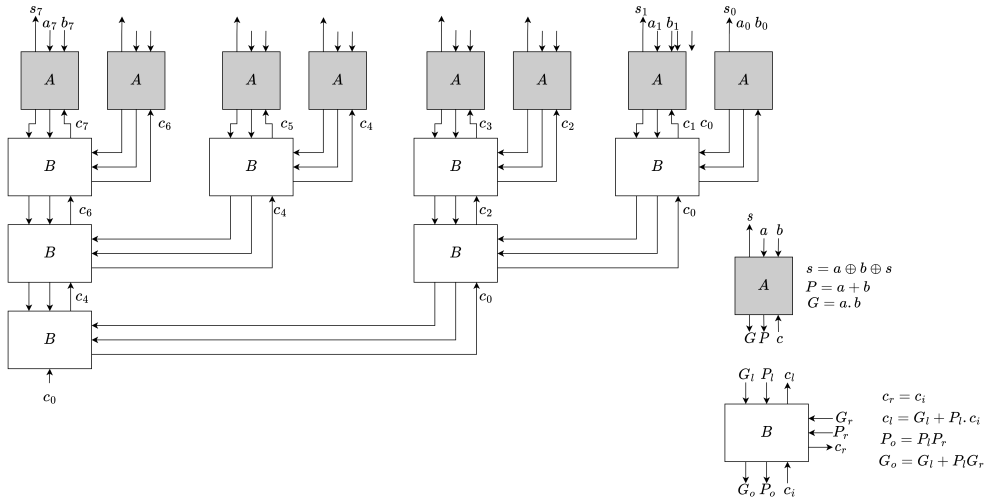

FIGURE 3.6 : Additionneur de forme "Carry look ahead"

- $P_i = (a_i + b_i)$ nommé le propagateur de la retenue

Il est crucial de calculer pour chaque position de chiffre si cette position va propager une retenue si elle provient de la droite. Ensuite, il faut combiner ces valeurs calculées afin de déduire, pour chaque groupe de chiffres, si ce groupe va propager une retenue.

Ainsi, la Figure 3.6 peut être élaborée en décomposant la propagation sous la forme d'un arbre dichotomique, ce qui permet de réduire la profondeur logique de la propagation de la retenue.

Gestion de la permutation Ce type d'additionneur favorise une propagation rapide de la retenue en s'appuyant sur l'architecture look ahead, enrichissant ainsi la dynamique de gestion des retenues. L'intégration de notre permutation à ce schéma nous mène à l'additionneur modifié illustré en Figure 3.7.

Ce type d'additionneur, semblablement au décalage, intègre une rétropropagation des éléments P et G facilitant le calcul de la retenue. La singularité du calcul de G réside dans la dépendance de la direction d'origine de l'entrée, soit de droite ou de gauche, la sélection s'effectuant toujours en accordance avec la clé K. Avec la propagation de P et G, l'évaluation des retenues devient réalisable. Les retenues, en suivant la logique dictée par la clé, se dirigent soit vers la gauche soit vers la droite. Lorsque $K = 0$, la retenue de poids le plus faible se déplace vers la droite, tandis que si $K = 1$, elle se dirige vers la gauche.

Pour effectuer la soustraction, on utilise souvent la technique du complément à deux. Le

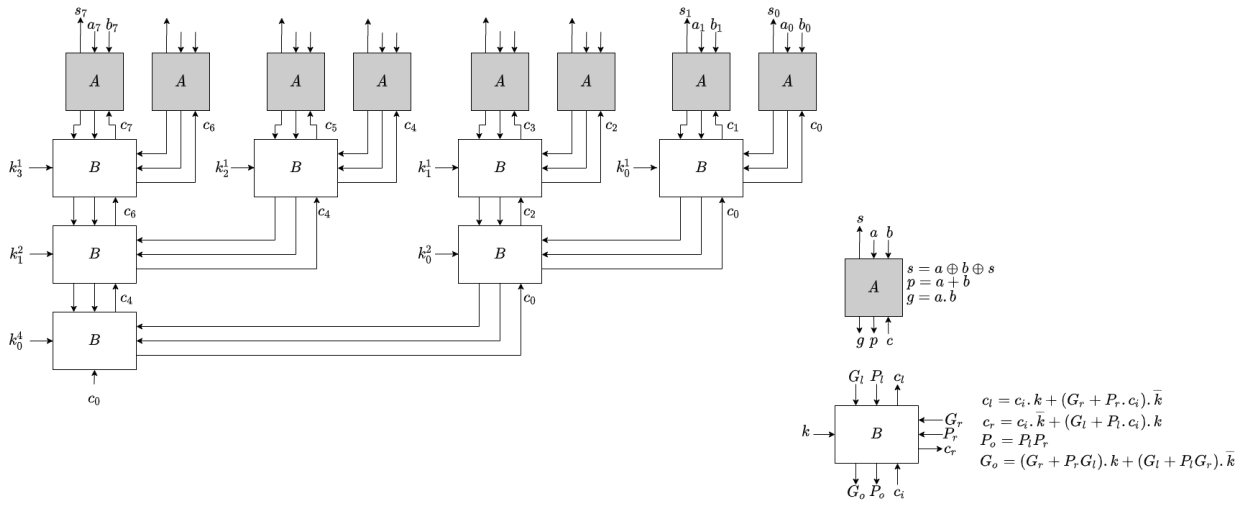

FIGURE 3.7 : Permute look ahead.

complément à deux d'un nombre est obtenu en inversant les bits du nombre et en ajoutant 1 au bit le moins significatif (LSB). Cependant, au lieu d'ajouter 1 au LSB après l'inversion des bits, on peut aussi introduire ce 1 via l'entrée de la retenue dans l'adder. Ainsi, pour effectuer la soustraction en utilisant un CLA, on configure l'entrée de la retenue à 1, et on présente le complément à un de l'une des entrées de l'adder. Le CLA effectue alors l'addition de, et l'entrée de la retenue de 1 permet de compléter l'opération de complément à deux, produisant ainsi le résultat correct de la soustraction.

3.4.3 Décalage

La seconde opération à traiter est le décalage. La méthode conventionnelle d'implémentation d'un décalage consiste à combiner des opérations de décalage. Cette combinaison s'effectue au moyen d'une structure comme illustré en Figure 3.8. La sortie du décalage 2^n est propagée si et seulement si un indicateur $Shift_{(2^n)}$ est présent. Ainsi, pour réaliser un opérateur de décalages, il est nécessaire de maîtriser les décalages de 2^n.

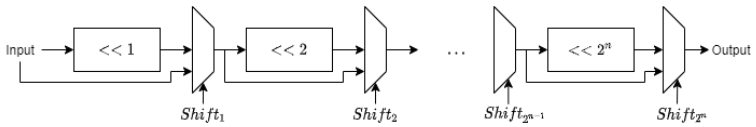

FIGURE 3.8 : opérateur de décalage.

Décalage à gauche de 1 bit Initialement, considérons le décalage à gauche de 1 bit. Pour illustrer cette permutation nous utiliserons la figure 3.9 qui expose la solution adoptée pour accomplir ce décalage de 1 bit à gauche sur un vecteur de 4 bits, avec en sortie cette donnée décalée toujours permutée avec la même clé. Ainsi, en entrée, une donnée $a_3a_2a_1a_0$ est permutée avec une clé de valeur 110, soit $Key^2 = 1$ et $Key^1 = 10$. Grâce à la segmentation en blocs de notre permutation, il est faisable d'exécuter le décalage à l'intérieur des blocs de taille 2. Effectivement, cette configuration assure que tous les bits dans un bloc de 2 bits sont consécutifs, il est donc requis d'inverser la position du bit de poids faible et de propager le bit de poids fort dans l'arborescence. Cette propagation s'effectue à l'aide de la clé.

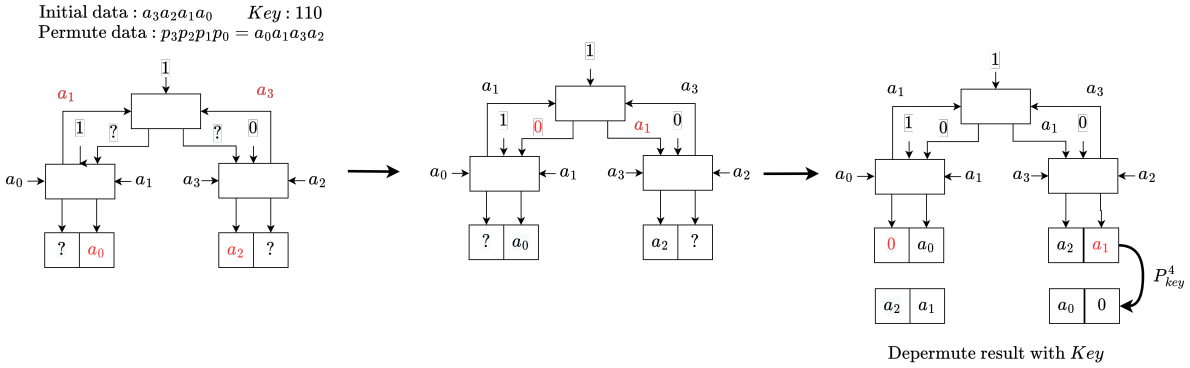

FIGURE 3.9 : Décalage de 1 bit à gauche d'une donnée sur 4 bits. La donnée est notée $a_3a_2a_1a_0$ Avec une clé $K = 110$, la permutation est $a_0a_1a_3a_2$ et la permutation décalé est $0a_0a_2a_1$..

Effectivement, l'enjeu réside dans la capacité à router les retenues afin de remplir les zéros laissés libres par le décalage.

En examinant de plus près la Figure 3.9, celle-ci dépeint la propagation des retenues. Initialement, le décalage au sein des blocs de deux bits est effectué et les retenues sont propagées aux étages supérieurs.

Parvenus aux étages supérieurs, l'opération d'inversion est réitérée ; en cas d'arrivée au dernier niveau, il est requis, pour les décalages logiques, de propager un 0 à travers notre schéma. La propagation du zéro à droite ou à gauche est dictée par la clé : si la clé est à zéro, les retenues sont correctement ordonnées, imposant ainsi la propagation du zéro vers la droite ; dans l'éventualité où la clé vaut 1, les retenues sont inversées, nécessitant dès lors la propagation du zéro sur la gauche. Une fois les retenues acheminées vers les blocs inférieurs, elles sont routées vers la sortie laissée libre. La sortie demeure ainsi judicieusement

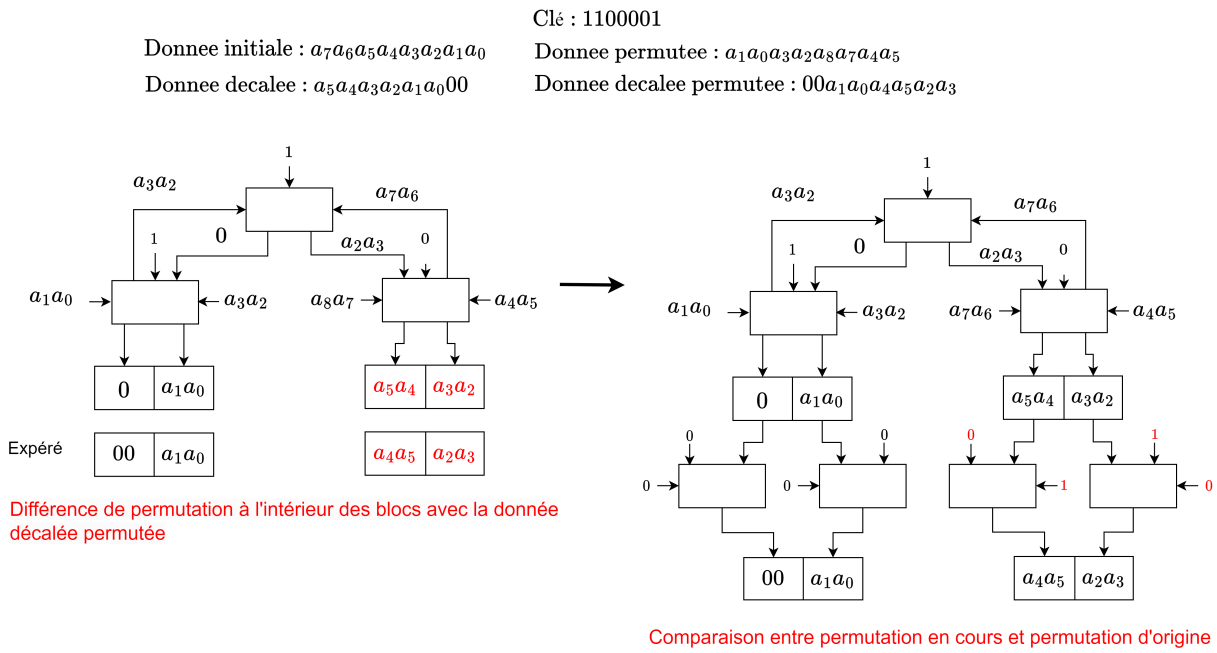

Figure 3.10 : Décalage de 2 bits sur une donnée 8 bits.

permutée conformément à la même clé que la donnée initiale.

Décalage à gauche de 2^n bit Les décalages par les puissances de 2 s'avèrent plus complexes à mettre en œuvre, bien qu'il soit aisé d'effectuer des décalages de blocs en suivant un schéma similaire au décalage de 1. Tandis que ce dernier déplaçait des blocs de 1 bit, le décalage de 2^n concerne des blocs de 2^n bits. Nous présentons en figure 3.10 un shift de 2 bits sur une donnée 8 bits. Si, dans un premier temps, on effectue un schéma analogue à un shift de 1 bit, on se rend alors compte qu'il faut non plus seulement décaler des blocs de bits, mais aussi permuter l'intérieur de ces blocs pour retrouver la permutation originelle. Pour effectuer cette permutation, il faut comparer la permutation en cours dans le bloc et la permutation originelle de la donnée en cas de différence on effectue la permutation comme dans la deuxième partie du schéma 3.10. Ainsi, pour effectuer un décalage 2^n, il faut deux structures : la première qui effectue le décalage de la donnée et réordonne l'intérieur des blocs, et une deuxième structure qui trouve la clé en cours dans les blocs pour savoir s'il faut les permuter ou non. Une solution est esquissée en Figure 3.11, distinguant la clé en cours Key_n^m et la clé selon laquelle le bloc est permuté $K_{D_n}^m$.

Structure de décalage des blocs La partie supérieure (blocs blanc et gris clair) du schéma illustre le décalage de la donnée, en parallèle avec la Figure 3.9. Nous traitons ici une donnée de 32 bits avec un décalage de 8 bits. Nous ne faisons plus l'inversion de deux bits, mais des blocs de 8 bits. Au terme de cette étape, nos blocs de 8 bits sont bel et bien décalés, bien que les valeurs à l'intérieur des blocs ne correspondent pas à la permutation suivant la clé.

Dans un second temps (blocs noirs), il est impératif de réarranger l'intérieur des blocs pour les harmoniser avec la permutation en cours. Pour ce faire, il convient de comparer Key_n^m, la clé de l'emplacement actuel du bloc, et $K_{D_n}^m$, la clé de l'emplacement précédent du bloc. Si elles diffèrent, le bloc est inversé ; dans le cas contraire, il demeure dans sa configuration d'entrée. Cette opération est reproduite pour tous les blocs constituant le bloc permuté.

Structure de clé en cours En fin de schéma, nous obtenons ainsi un bloc décalé de 8 bits et permuté conformément à la clé d'entrée. Toutefois, pour réaliser le réordonnancement des blocs, il est impératif de récupérer les permutations de chacun des blocs. Un schéma analogue au décalage de 1 est employé à cet effet. Cette portion est illustrée dans la partie supérieure de la Figure 3.12, dépeignant un décalage de 8 dans un contexte de permutation $P_{key^3 2}$. Les entrées Key_i représentent la clé englobant toutes les sous-clés dépendantes de ce bloc, par exemple :

$$key_0^4 key_1^2 key_0^2 key_3^1 key_2^1 key_1^1 key_0^1$$

En dernier lieu, la partie inférieure concerne la mise à jour des permutations internes au bloc en fonction de la clé en cours. Chaque divergence entre la clé en cours et la clé du bloc induit une permutation sur les clés des blocs le constituant. Si une différence est observée entre la clé en cours et la clé du bloc, les positions des clés des blocs constituant le bloc sont inversées.

Avec ces structures, il est effectivement possible de réaliser tous les décalages envisageables grâce à la combinaison des décalages de 2^n. Pour chaque décalage, l'équilibrage du nombre d'étages entre la permutation des blocs et le réordonnancement interne des blocs variera, néanmoins la profondeur globale du décalage demeurera constante à $log_2(\text{len})$, où len représente la taille de la donnée.

Autres décalages Le décalage à droite implique l'inversion du sens de toutes les permutations effectuées lors du décalage à gauche. Il est notable que ces modifications consistent uniquement en l'obtention du complémentaire de la clé. Effectivement, le fait de prendre

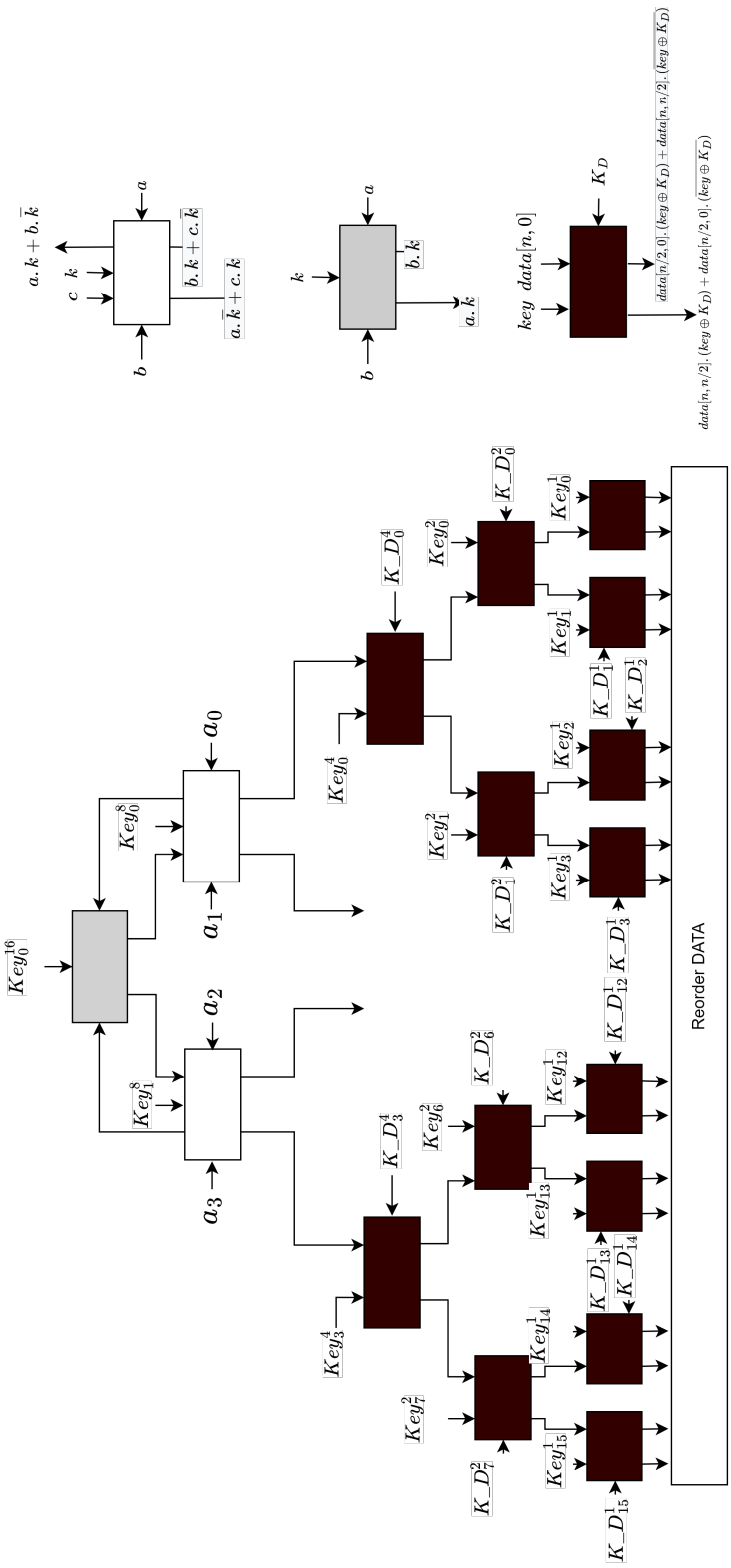

Figure 3.11 : Décalage de 8 bits sur une donnée 32 bits.

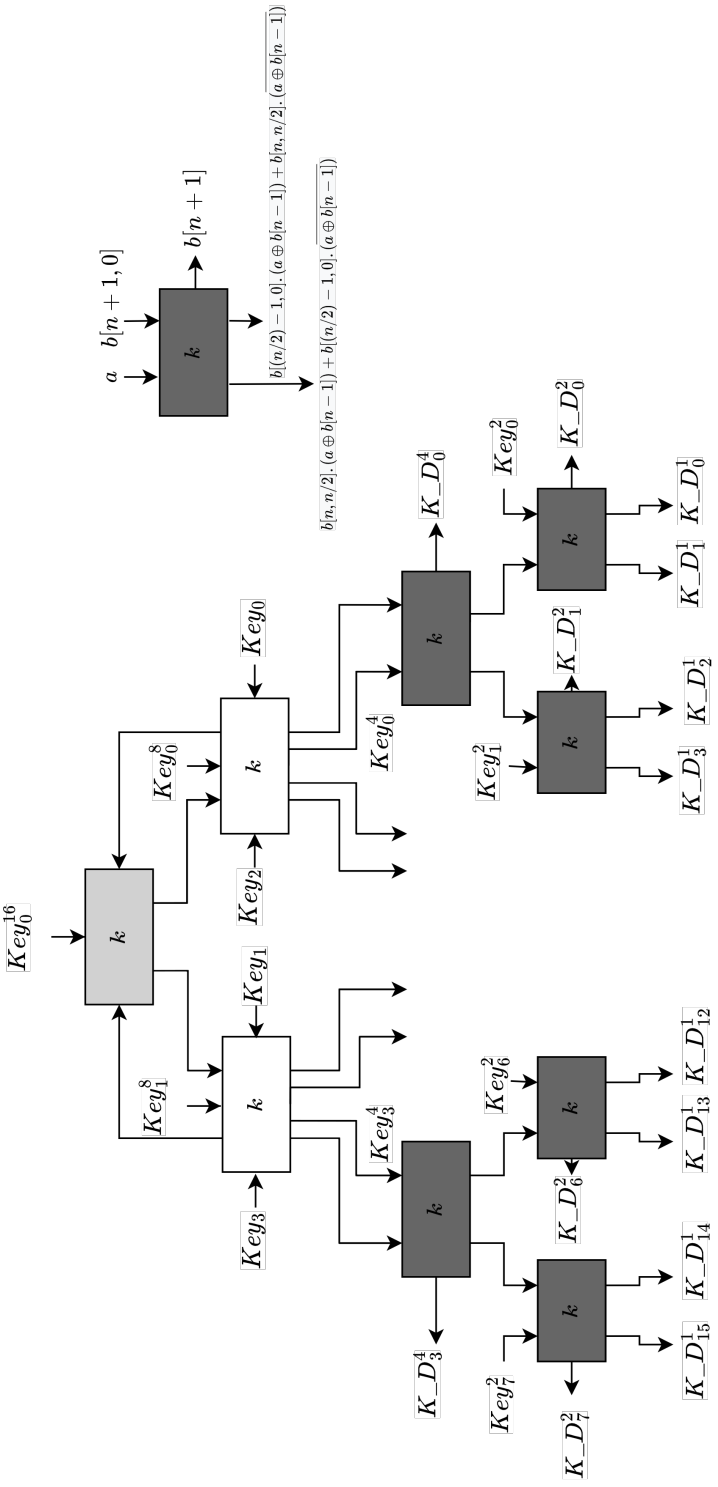

Figure 3.12 : Retrouver les clés en cours.

le complémentaire de la clé exécute toutes les transpositions dans le sens inverse.

Quant au dernier type de décalage, la rotation, seul le premier bloc requiert une modification. Il n'y a plus de notion de mise à zéro ou de remplacement de bit, les bits atteignant le sommet de l'architecture doivent inévitablement être inversés, car ce qui sort de la rotation doit être réinjecté dans son entrée. Ainsi, l'inversion est inconditionnelle et ne dépend pas de la clé.

Pour réaliser un décalage arithmétique à droite, il est à noter que le décalage arithmétique à gauche est équivalent au décalage logique à gauche. L'exécution de ce décalage nécessite initialement de retrouver le bit de poids fort non permuté, qui peut être rapidement localisé via une sélection des sous-blocs le contenant. Une fois le bit de poids fort identifié, le décalage peut être effectué. La seule distinction réside dans la duplication du bit de poids fort dans les emplacements laissés vacants par le décalage. Ces espaces libres sont générés dans le premier bloc de l'architecture, le bloc gris clair, où la mise à zéro doit à ce stade être remplacée par le bit de poids fort.

Tous les blocs sont identiques, car le bit de poids fort est celui situé tout à gauche, donc dans tous les blocs internes ce bit doit être déplacé à gauche quand la clé est à zéro. Ce schéma reproduit simplement la séquence de blocs permettant de retrouver l'emplacement du bit ; pour le bit de poids fort, il est toujours à gauche, raison pour laquelle tous les blocs sont identiques.

3.4.4 Multiplication

La dernière opération à réaliser est la multiplication. Une panoplie de méthodes de multiplication est disponible, chacune avec ses avantages et inconvénients. La méthode itérative, bien qu'élémentaire, requiert un nombre conséquent de cycles, toutefois, diverses optimisations peuvent être envisagées.

Méthode itérative La méthode itérative est l'approche homologue à une multiplication posée, incarnant une suite d'additions où le multiplicande est décalé à chaque palier, comme illustré en figure 3.13. À chaque étape, le produit partiel est incorporé à la somme finale, engendrant un processus qui, bien qu'ordonné, demande 32 cycles pour achever une multiplication sur 32 bits. Cette méthode a l'avantage d'être simple et ne nécessite que des opérations de décalages et d'additions. Bien que rudimentaire, cette méthode est un choix fiable pour des systèmes ne pouvant accommoder des méthodes de multiplication plus

complexes ou optimisées. Elle présente une voie claire et méthodique pour accomplir la multiplication, tout en s'accommodant aisément à la structure de permutation en décalant et en additionnant conformément à la clé de permutation. Pour la réalisation matérielle

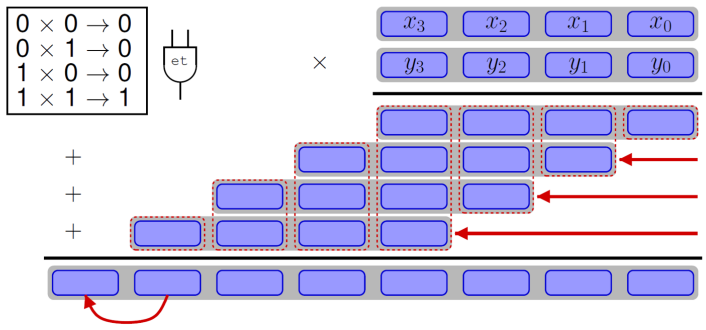

FIGURE 3.13 : Multiplication binaire.

de ce type de composant, l'exigence en termes d'éléments est minime : un registre de 32 bits, un additionneur et un registre à décalage de 64 bits suffisent. La structure générale est dépeinte en figure 3.14 et l'algorithme en 1 . La multiplication envisagée est sur n bits. À l'initialisation, le multiplicande est logé dans le registre X et le multiplieur dans le registre Y_0. Puis, à chaque cycle d'horloge, en fonction du bit de poids faible $Y_0[0]$ du registre Y_0 :

- si égal à 1, le multiplicande est additionné à Y_1, le résultat est stocké dans Y_1 et le débordement dans C,
- si égal à 0, aucune action n'est entreprise.

S'ensuit un décalage à droite sur les 32 bits (avec C concaténé à Y_1 et Y_2). Après n cycles, les bits de poids forts (Most Significant Bits, MSB) sont accessibles dans le registre Y_1 et les bits de poids faibles (Less Significant Bits, LSB) dans Y_0.

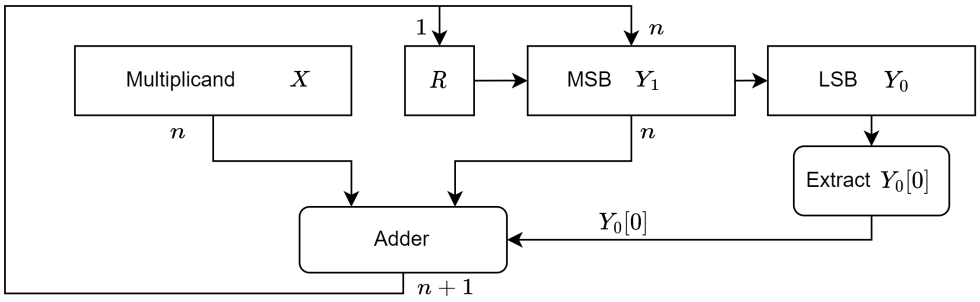

FIGURE 3.14 : Description matérielle d'une multiplication itérative sur n bits.

Algorithm 1: Iterative multiplication

Data: $X, Y_0 \in GF(2^n)$
Result: $Y_1||Y_0$ with $Y_1, Y_0 \in GF(2^n)$ ▷ || signifie concaténation

initialization
$Y_1 \leftarrow 0, C \leftarrow 0, Counter \leftarrow 0$
while $Counter \leq n$ **do**
  if $Y_0[Counter] = 1$ **then** ▷ nécessite de trouver la position d'un bit sur des données permutées
    $C||Y_1 = X + Y_1$ ▷ nécessite une addition permutée
    $C||Y_1||Y_0 >> 1$ ▷ nécessite un décalage d'un bit et 2 permutations
  else
    $C||Y_1||Y_0 >> 1$ ▷ nécessite un décalage d'un bit et 2 permutations
  end
  $Counter = Counter + 1$
end
return $Y_1||Y_0$

Pour rendre cette description compatible avec les permutations, quelques ajustements mineurs sont nécessaires. En premier lieu, il est impératif de remplacer l'additionneur par celui détaillé en 3.4.2. En outre, le registre à décalage doit être substitué par celui exposé en 3.4.3. Cependant, certaines subtilités doivent être intégrées dans cette implémentation. Il ne s'agit pas ici d'un décalage conventionnel où un zéro est inséré pendant le décalage, mais il est nécessaire de prendre la retenue de l'addictionneur. Pour ce faire, le sommet de l'arbre doit être modifié comme illustré en figure 3.15. De plus, une structure doit être ajoutée pour sélectionner le bit de poids faible dans une donnée permutée, et pour ce faire, il est conseillé d'utiliser une structure arborescente qui, assistée par la clé, sélectionne la branche hébergeant le bit de poids faible.

Optimisation de Booth L'algorithme de Booth permet la multiplication d'entiers binaire dans une représentation en complément à 2 en éliminant certaines additions par des soustractions. En effet, quand un bit 0 dans le multiplieur apparaît, l'addition n'est pas effectuée mais seulement le décalage. De même, une suite consécutive de 1 du bit 2^k au bit 2^m peut être vu comme la soustraction de $2^{k+1} - 2^m$ qui n'introduit que deux 1 (et donc une addition et une soustraction) et potentiellement beaucoup de 0 qui eux n'impliquent que des décalages. Le nombre de cycle est ainsi allégé d'un facteur 2 (Cf l'algorithme de Booth 2. La principale différence dans l'implémentation matérielle est le calcul du complément à deux sur la donnée permutée. On effectue pour cela une inversion et un additionneur permuté.

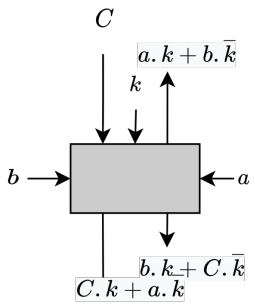

Figure 3.15 : Bloc de décalage pour la multiplication itérative.

Pour obtenir les deux bits de poids faible, il faut utiliser là aussi une structure qui sélectionne les bonnes branches. Il faut cependant les réordonner une fois qu'on les a obtenu. De plus, le bit Y_{-1} est retenu en sortie de décalage du cycle précédent qui est maintenant de deux bits vers la droite. Enfin, l'additionneur sélectionne s'il doit faire une addition, une soustraction ou ne rien faire selon les valeurs des bits en entrée.

Algorithm 2: Booth's multiplication

Data: $X, Y_0 \in GF(2^n)$
Result: $Y_1||Y_0$ with $Y_1, Y_0 \in GF(2^n)$ ▷ || signifie concaténation

initialization
$Y_1 \leftarrow 0$, $Counter \leftarrow n$
while $Counter \geq 0$ **do**
  if $Y_0[0]||Y_{-1} = 10$ **then**
    $Y_1 = Y_1 - X$ (add if X and Y_1 are 2-complemented)
  end
  if $Y_0[0]||Y_{-1} = 01$ **then**
    $Y_1 = Y_1 + X$
  end
  2 bits arithmetic shift right of $Y_1||Y_0||Y_{-1}$
  $Counter = Counter - 1$
end
return $Y_1||Y_0$

L'implémentation matérielle est donnée en figure 3.16. La principale différence est le calcul du complément à deux sur la donnée permutée. Pour obtenir les deux bits de poids faible, il faut qu'on utilise là aussi une structure qui sélectionne les bonnes branches. Il faut cependant les réordonner une fois qu'on les a obtenus. De plus, le bit Y_{-1} est retenu en sortie

de décalage du cycle précédent qui est maintenant de deux bits vers la droite. Enfin l'adder sélectionne s'il doit faire une addition, une soustraction ou ne rien faire.

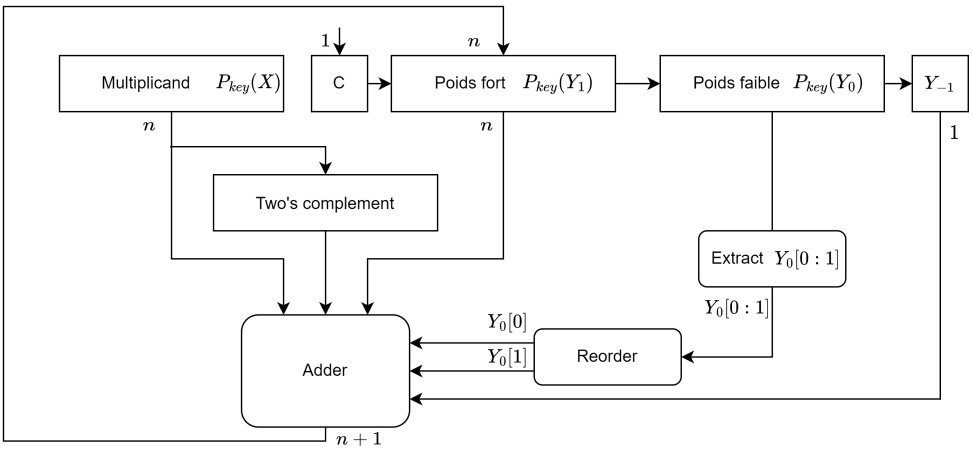

Figure 3.16 : Multiplication de booth avec des données permutées

Optimisation possible Il est envisageable d'explorer d'autres méthodes d'optimisation pour réduire davantage le nombre de cycles, comme l'application du Booth avec un radix plus élevé, ou encore des méthodes de réduction rapide des sous-produits à l'aide d'arbres de Dada [Dad65]. Les méthodes employant les arbres de Dada engendrent la problématique de calculer tous les sous-produits, et donc d'exécuter de nombreux décalages, ce qui induit un coût significatif avec notre solution. Cependant, des structures de décomposition dichotomique, assistées par la méthode de Karatsuba [KO63], peuvent constituer une voie prometteuse pour effectuer des multiplications rapides avec des données permutées.

3.4.5 Performance

Le coût associé à chaque opérateur est explicité dans le tableau 3.1. Il est observé que le nombre de portes requis pour la permutation est relativement modéré, de même que pour l'addition, pour laquelle l'intégration astucieuse de la permutation au sein de la structure du carry look ahead s'avère plus efficace que la simple incorporation d'une permutation. Néanmoins, bien que réalisé en un seul cycle, le décalage engendre un surcoût considérable, nécessitant des recherches sur des optimisations de son implémentation.

module	Area (GE)
32-bit permutation	2013
32-bit Carry Look Ahead Adder	1806
32-bit Permuted Carry Look Ahead Adder	3015
32-bit Barrel Shifter	4976
32-bit Permuted Barrel Shifter	16596

Table 3.1 : Performances of permuted operators compared to canonical ones in terms of Area (GE).

3.4.6 Masquage

La seule contremesure prouvée sûre contre les attaques SCA et le maintien de la confidentialité dans le cadre de notre modèle d'attaquant reste les différentes formes de masquage et en particulier le masquage booléen et son probing model [ISW03b]. Par contre, la permutation associée à la donnée est une redondance qui comme telle contribue à augmenter les fuites par canaux auxiliaires. La clé de la permutation fuit relativement peu dû au fait que les portes de Fredkin montre une parfaite symétrie par rapport au bit de clé et à son complémentaire. nous avons l'équivalent d'une protection de type dual rail comme le WDDL [TV04] qui toutefois n'est que théorique car encore sensible aux dispersions de placement-routage et aux glitchs. Pour pallier ces vulnérabilités, la propriété d'homomorphisme par rapport au XOR peut être mise à profit pour masquer la donnée et la donnée permutée sans remettre en cause la vérification du calcul qui peut rester masquée comme le montre la Figure 3.2.

Nombre de masque Étant donné que donnée et donnée permutée sont constituées des mêmes bits mais à des places différentes dans le mot, il est envisageable sans compromettre la sécurité d'utiliser le même masque. Cette approche tout en facilitant l'étape de vérification ne nécessite que le stockage d'un seul masque permettant une réduction significative du surcout matériel de la solution. Par contre, elle présente des inconvénients. Comme pour les opérations arithmétiques, les opérations canoniques et permutées sont différentes, elles vont nécessiter des bits aléatoires différents et par conséquent générées en fin de calcul des masques différents. Une autre faiblesse est que l'on perd l'intégrité du calcul. En effet, une faute sur le masque commun ne permet pas de s'assurer de l'intégrité sur la donnée d'origine, la vérification devant rester masquée. Malgré le surcout, nous avons choisi d'avoir des masques aléatoires différents et de suivre une vérification des calculs entièrement masquée telle que montrée dans la Figure 3.2.

Schéma de masquage La quête du masquage d'une Unité Arithmétique et Logique (ALU) représente un idéal recherché par la communauté scientifique depuis près de deux décennies, un voyage parsemé de nombreux revirements. Cette réalisation promet d'ouvrir les horizons vers l'élaboration de logiciels cryptosystémiques sans la menace constante des fuites de canaux auxiliaires. La contremesure présentés ne visent pas à introduire un nouveau schéma de masquage, ni à évaluer la sécurité du schéma choisi (tâche accomplie dans les publications séminales), mais à illustrer les contraintes et l'investissement requis pour leur incorporation au sein d'une ALU standard et d'une ALU permutée. Ils visent également et surtout à démontrer que les schémas et les ressources nécessaires sont désormais à notre disposition pour concrétiser cet idéal.

Au fil des années, une multitude de variantes de masquage matériel ont été suggérées [ISW03b] [NRR06] [GMK17] [GM18], visant à réduire le coût d'implémentation via une diminution de l'encombrement spatial et une réduction du nombre de bits aléatoires requis [GMK16][Sug19]. Ces initiatives aspirent également à renforcer l'efficacité sécuritaire en affinant de manière formelle les exigences de protection contre les glitches et les transitions [MPG05], ainsi qu'en termes de composition de portes sécurisées, souvent désignées sous le terme de gadgets (principalement des portes AND et XOR sécurisées). Une première approche de composabilité de gadgets masqués, nommée "Non-Interférence" (NI) [Bar+15], repose sur le principe de « simulation parfaite de la distribution de probabilité conjointe ». Toutefois, cette définition néglige l'effet de propagation de sonde, révélant la nécessité de considérer, au niveau d'une sonde, tous les signaux contribuant à sa formation en amont. La notion de String Non-Interférence (SNI) [Bar+16b] soutient alors que cette propagation de sonde doit être entravée par l'ajout d'un bit aléatoire. Plus récemment, il a été démontré que cette définition était insuffisante pour garantir la composition de gadgets, donnant lieu au concept de Probe Isolation Non-Interférence (PINI) [CS20]. Ce dernier énonce qu'une sonde ne peut se propager que dans son propre domaine de partage. Ainsi, la composition de gadgets PINI demeure conforme à la notion de PINI.

Des avancées récentes visent à minimiser cette latence. Il est notable de mentionner la Logique LUT based masked dual rail (LMDPL) [LMW14] qui emploie du dual rail (un signal booléen est toujours associé à son complémentaire) et qui a permis de réaliser une Sbox AES masquée au premier ordre, éprouvée contre les glitchs avec une latence extrêmement réduite. Cependant, cette solution ne confère pas au concepteur la maîtrise sur le pipelining du circuit et ne se montre pas compatible avec des implémentations d'ordres supérieurs. L'association du dual rail avec des fonctionnalités fréquemment rencontrées dans la

conception asynchrone a conduit récemment à un schéma satisfaisant. Le SESYM, ou Self Synchronized Masking, substitue les étages de registres [Nag+22] par un encodage partiel en dual rail des signaux masqués, assurant ainsi la synchronisation au sein du circuit. Les deux schémas évoqués présentent des mérites distincts. Le premier, dénommé par l'acronyme LMDPL, se singularise par son faible surcoût matériel, surtout que la configuration dual rail n'est pas essentielle pour qu'il soit catégorisé comme un schéma PINI. Le second schéma, SESYM, se distingue par sa latence réduite, un trait bénéfique dans des scénarios où la célérité de traitement est recherchée.

3.4.7 Sécurité

Dans cette section, nous discutons de la sécurité de la permutation contre les attaques en faute et les attaques canaux auxiliaires. La structure choisie de notre solution permutée en Figure 3.2 montre que nous avons cherché à décorreler les deux types de contremesures : la duplication avec l'ajout d'une permutation contre les fautes et le masquage appliqué aux deux domaines contre les fuites side channel. L'atout qui permet cette facilité est l'homomorphisme de la permutation par rapport à l'opérateur logique XOR.

3.4.8 Protection contre les canaux auxiliaires

La conservation de la confidentialité de la donnée dans le modèle d'attaquant incapable de faire une intrusion physique comme du micro-probing dans le circuit se limite à la gestion des fuites canaux auxiliaires. La contre-mesure avancée est donc le masquage. Aucune évaluation du masquage implémenté n'a été faite car une force de notre approche est que tous les types de masquage composables PINI sont compatibles avec notre permutation. Par conséquent, il convient au concepteur de juste s'assurer que les gadgets utilisés ont montré les protections attendues ce qui est généralement évalué dans les articles originaux des schémas de masquage choisis et de générer son circuit avec des outils vérifiant formellement l'implémentation comme AGEMA [Kni+21]. Par ailleurs, il faut aussi remarquer que même sans masquage, la permutation en utilisant des portes de Fredkin a une implémentation symétrique par rapport à ses entrées et à leur complémentaire limitant les fuites au minimum par construction.

3.4.9 Protection contre les fautes

La permutation définie présente au moins les propriétés d'une duplication contre les attaques en faute. Ainsi, dans le cas de fautes stochastique, l'apport par rapport à une duplication n'est pas pertinent. Par contre, contre un adversaire cherchant à cibler des bits de la donnée bien précis avec par exemple un laser pouvant éventuellement focalisé plusieurs faisceaux à des temps différents, l'aléatoire introduit par la clé secrète complique lourdement la tâche de l'attaquant. Toutefois, il faut remarquer que la permutation utilisée engendre une certaine localité dans le mélange des bits de la donnée. En effet, si elle permet de positionner un bit à n'importe quelle position, les bits qui étaient à l'origine proche de celui-ci vont se retrouver à une position également assez proche dans le domaine permuté.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une permutation de bits de données simple visant à améliorer efficacement l'intégrité des calculs effectués par l'ALU. Cette permutation est conçue de manière à être homomorphe par rapport aux opérations logiques et arithmétiques, simplifiant ainsi le processus de vérification. En utilisant une clé secrète qui peut être régulièrement mise à jour, nous sommes en mesure de contrer les attaques au laser qui pourraient cibler spécifiquement des bits individuels, introduisant ainsi une forme de polymorphisme matériel léger dans l'implémentation.

La présence de cette clé secrète garantit également l'authenticité en temps d'exécution, et ce, non seulement au démarrage du système, comme cela est couramment fait dans les procédures de démarrage sécurisé, mais également tout au long du fonctionnement du système. Les résultats ont montré une empreinte spatiale relativement modeste pour l'opération de permutation (2013 GE) et l'opération d'addition permutée (3015 GE). En revanche, l'opération de décalage de bits s'est avérée plus complexe et a nécessité davantage de ressources, soit 16596 GE.

Il est important de souligner que l'objectif de ce travail était de concilier trois aspects cruciaux de la sécurité, à savoir l'intégrité, l'authenticité et la confidentialité, ce qui représente un défi majeur. La nature homomorphe de la permutation, ainsi que sa conception légère, ont conduit à l'idée de masquer les opérations permutées afin de prévenir les fuites d'informations par le biais de canaux secondaires.

4

Chemin d'instruction et de contrôle

Ce chapitre aborde les vulnérabilités et les mesures de défense associées au chemin de contrôle du pipeline. Deux contributions sont présentées. La première vise à sécuriser le chemin contre les attaques par canaux auxiliaires et par injections de fautes, depuis la récupération des instructions jusqu'à leur décodage, tout en minimisant les surcoûts associés. La seconde propose une nouvelle approche pour la vérification de l'intégrité des données de contrôle, en exploitant la parité pour tirer avantage des limites des méthodes d'injections de fautes.

4.1 Modèles d'attaquants

Les attaques physiques et par observation se multiplient, en particulier contre les processeurs destinés à l'Internet des Objets (IoT), et les contre-mesures habituellement proposées induisent souvent un surcoût significatif. Le chemin de données est particulièrement vulnérable aux attaques par injections de fautes, étant donné qu'il constitue le noyau de la gestion des instructions. Les injections de fautes, notamment celles provoquant des sauts d'instruction [Men+20; Dut+19], peuvent contourner de nombreuses protections algorithmiques. Ces attaques peuvent également altérer les instructions pour modifier soit le flux d'exécution, soit le résultat des calculs. Les perturbations dans le flux d'exécution peuvent notamment affecter les prises de branchement ou permettre des sauts arbitraires dans le code. Outre ces menaces, la propriété intellectuelle du code source peut également être compromise, par exemple via le désassemblage utilisant des canaux auxiliaires [CLH19].

Bien que d'autres formes d'attaques existent, telles que les attaques par cold boot [GM13] ou les attaques logicielles comme l'écrasement de l'adresse de retour sur la pile, ces menaces sortent du cadre de notre modèle d'attaquant. Pour contrer ces types de menaces, des techniques de chiffrement et de contrôle de flux d'exécution sont généralement les plus appropriées.

4.2 Contre-mesures existantes

Pour garantir la sécurité des instructions face aux menaces identifiées, quatre propriétés essentielles doivent être respectées :

- L'instruction en cours d'exécution doit être logiquement précédée par l'instruction antérieure dans le flux d'exécution, afin d'éliminer les sauts et modifications non autorisées d'instructions.
- La phase de décodage (DECOD) doit être sécurisée pour s'assurer qu'elle s'effectue sans erreurs.
- Les branchements doivent être protégés, garantissant que le processeur exécute la branche correcte du code.
- La valeur de chaque bit de l'instruction ne doit pas être directement accessible, afin de prévenir les fuites d'information.

Dans l'état actuel de la recherche, ces quatre propriétés ne sont jamais simultanément et efficacement adressées. Les unités de vérification de l'intégrité du flux de contrôle (CFI) [CV17] sont les solutions les plus couramment employées pour la protection du flux d'exécution. Elles peuvent assurer l'intégrité des instructions jusqu'aux premiers stades du pipeline [Wer+18] et peuvent également prévenir les fuites d'information par des canaux secondaires dans la hiérarchie de la mémoire, si des méthodes de cloisonnement sont employées [SEH20]. Néanmoins, ces mesures ne couvrent pas la phase de décodage des instructions ni les branchements, et elles induisent un surcoût significatif.

Dans les architectures de processeurs robustes, telles que Klessydra [Bla+19], une duplication spatiale de l'étage DECOD est mise en œuvre pour sécuriser le décodage. De plus, la duplication temporelle est utilisée pour prévenir les sauts d'instruction [Mor+14]. Toutefois, ces solutions basées sur la duplication entraînent un surcoût matériel ou temporel considérable.

Pour contrer les attaques par canaux auxiliaires, les techniques de masquage sont fréquemment utilisées. Cependant, ces méthodes engendrent des coûts d'implémentation élevés, notamment en raison de la nécessité de doubler la taille des instructions pour stocker le masque.

4.3 SECDEC

Notre contribution consiste à proposer deux solutions étroitement liées, mais avec des implications d'implémentation distinctes, pour assurer la sécurité du chemin d'instruction contre les attaques par perturbation et observation. La première solution présente un surcoût matériel et une augmentation négligeable de la taille de la mémoire d'instruction, mais elle induit une dépendance du code compilé vis-à-vis de contraintes microarchitecturales spécifiques. La deuxième solution, en revanche, est exempte de ces contraintes microarchitecturales, mais nécessite une redondance dans la phase de décodage des instructions.

Ces deux approches requièrent des modifications au niveau du backend du compilateur. Toutefois, ces ajustements sont relativement simples à implémenter et peuvent être adaptés à diverses stratégies de compilation.

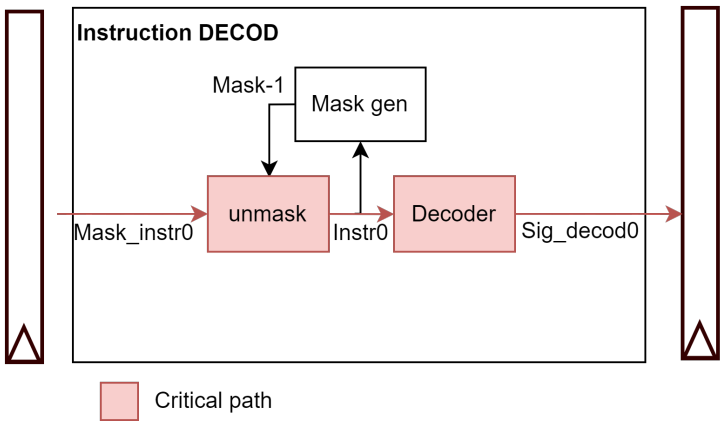

FIGURE 4.1 : Pré-décodeur.

4.3.1 Présentation du fonctionnement

Description de la contremesure

Cette contribution vise à aborder les quatre propriétés de sécurité énoncées précédemment en exploitant les signaux générés par l'instruction précédente pour masquer l'instruction en cours d'exécution. Il s'agit d'un masquage booléen non aléatoire, étant donné qu'il est déterminé par l'instruction précédente. Concrètement, un masque est généré au cycle N et est ensuite utilisé lors de l'étape de décodage (DECOD) du cycle $N + 1$ pour démasquer l'instruction entrante. Une option consiste à prendre les signaux d'entrée du décodeur, comme illustré à la Figure 4.1. Dans ce cas, c'est l'instruction elle-même qui sert à la génération du masque. Nous désignons cette solution comme étant le pré-décodeur.

Il est également envisageable de capter les signaux de sortie du décodeur, comme le montre la Figure 4.2. Cette variante est désignée sous le terme de post-décodeur.

Dans cette configuration, l'instruction doit être masquée dès son arrivée à l'étape DECOD. Pour mettre en œuvre ce masquage, des passes de compilation supplémentaires sont nécessaires et doivent être intégrées dans le backend du compilateur. Par conséquent, chaque instruction est intrinsèquement liée à celle qui la précède. Si une anomalie se produit au cycle N, le démasquage de l'instruction lors de l'étape de décodage au cycle $N+1$ sera différent de celui en l'absence de défaut. Cette méthode permet donc de s'assurer que l'instruction courante est exécutée immédiatement après l'instruction précédente dans le code machine, éliminant ainsi les risques de sauts ou de modifications non autorisées d'instructions.

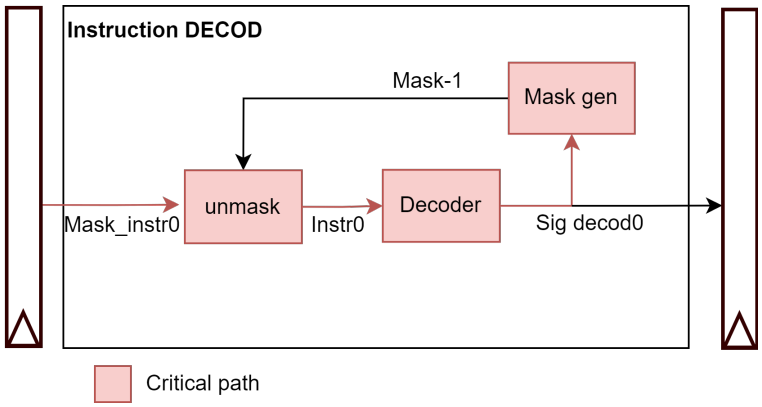

Figure 4.2 : Post-décoder.

En cas de branchement, deux chemins d'exécution sont possibles. Il est fréquent, lors de campagnes d'injection de fautes, de tenter de forcer le processeur à emprunter la mauvaise branche afin d'exécuter du code malveillant. La solution proposée permet de distinguer les deux branches en attribuant des masques différents en fonction de la branche effectivement prise, et ce, en se basant sur les signaux de décodage. Ainsi, contrairement aux mécanismes de CFI existants, qui sont généralement vulnérables au moins lors de l'étape de décodage, la validité du branchement est garantie tout au long du chemin d'instruction.

Le choix définitif de la branche à suivre est arrêté lorsque les conditions de branchement sont évaluées, ce qui se produit généralement à la fin de l'étape EXEC. La seule manière de forcer le processeur à prendre la mauvaise branche serait de corrompre l'étape EXEC. Ce cas est traité par les mécanismes de tags d'intégrité, comme présenté en section 3.3.

La détection d'erreurs repose sur les anomalies de décodage induites par une faute. Une erreur sur l'instruction à décoder, ou sur son masque, entraîne une modification du décodage. Si ce décodage est invalide, une exception "instruction invalide" est levée, permettant ainsi la détection de l'erreur. Toutefois, même si l'instruction défectueuse est valide (c'est-à-dire qu'elle fait partie du jeu d'instructions), le masque généré par cette instruction sera différent de celui de l'instruction qui aurait dû être exécutée en l'absence de faute. Par conséquent, l'efficacité de cette méthode de détection est accrue lorsque le jeu d'instructions est clairsemé, c'est-à-dire qu'il comporte un grand nombre d'instructions invalides par rapport au nombre d'instructions valides.

Si une proportion significative des opcodes est invalide, la détection de l'erreur ne prend que quelques cycles. La Figure 4.3 illustre la propagation d'une faute à travers les différentes étapes : extraction de l'instruction (IF), décodage de l'instruction (DEC), et exécu-

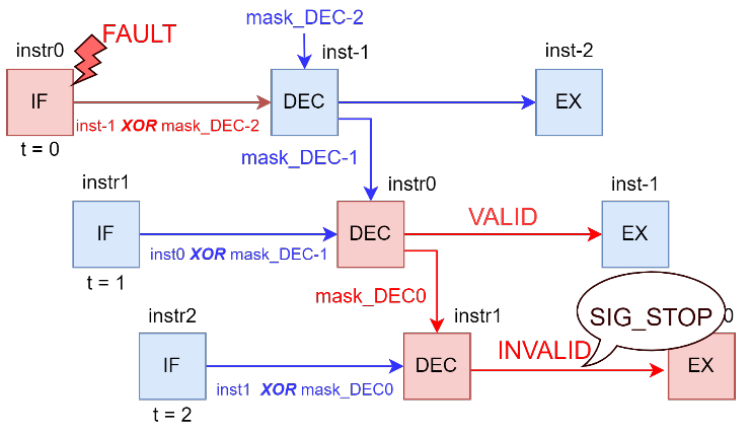

FIGURE 4.3 : Propagation et détection d'une instruction fautées.

tion (EXEC) – d'un processeur généraliste.

Ainsi, si une faute se produit au temps $t = -1$, elle génère un masque erroné qui est propagé au cycle suivant, sans toutefois déclencher de détection d'erreur à l'étape DECOD. Il en est de même pour l'étape DECOD au cycle 0. Cependant, au cycle 1, l'étape DECOD détecte une instruction invalide et lève une exception "instruction invalide", ainsi qu'un signal d'arrêt correspondant. Dans l'exemple de la Figure 4.3, le processeur prend donc 2 cycles pour détecter la faute.

Pour élaborer une stratégie de détection de fautes efficace, il est crucial d'évaluer quels segments de l'instruction sont les plus susceptibles de permettre une détection rapide des erreurs. Comme représenté à la Figure 4.4, différents types d'instructions existent, et les champs susceptibles de contribuer à des erreurs de décodage sont les suivants : opcode, funct3 et funct7. En outre, les bits de registres (rd, rs1, rs2), qui correspondent soit aux registres soit aux valeurs immédiates, sont invariablement valides, rendant leur utilisation inadéquate pour la détection d'erreurs. Par conséquent, les bits d'opcode, complétés dans une moindre mesure par les bits de funct3 et funct7, seront privilégiés pour la détection des erreurs de décodage. Il convient toutefois de noter que ces bits ne permettent pas de détecter les fautes pour tous les types d'instructions.

Cette contre-mesure a été implémentée dans un processeur CV32E40P RISC-V, mais l'approche est aisément transposable à d'autres ensembles d'instructions (ISA) ou architectures.

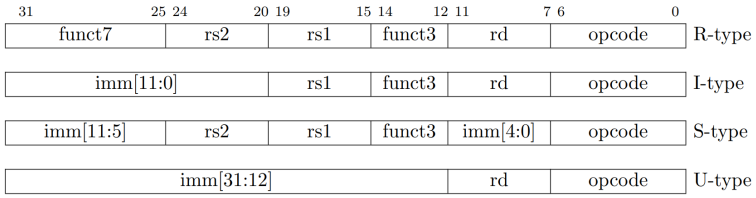

Figure 4.4 : Types d'instructions RISC-V.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Table 4.1 : Tableau de la Sbox de Piccolo.

Choix du masque

Pour optimiser la rapidité de la détection, c'est-à-dire pour que toute faute entraîne le plus rapidement possible une instruction invalide, deux approches complémentaires peuvent être envisagées. La première consiste à instaurer une interdépendance entre les bits. Si une faute se produit, elle doit entraîner un maximum de modifications sur les bits susceptibles de provoquer une erreur de décodage, en particulier les bits d'opcode et, dans une moindre mesure, les bits de funct3 et funct7. La seconde approche concerne la propagation des fautes à chaque nouveau cycle.

Pour une détection efficace, si une modification survient, le masque généré doit être substantiellement différent du masque légitime, tout en accordant une importance accrue au bit d'opcode. À cette fin, nous générons deux masques en utilisant la S-box à 4 bits du chiffrement léger Piccolo [Shi+11], présentée dans le tableau 4.1 et la figure 4.5, en raison de sa faible empreinte matérielle. Le premier masque utilise des blocs de 4 bits consécutifs pour créer l'interdépendance, tandis que le second masque prend 4 bits aux indices $i, i+8, i+16, i+24$ avec $i \in [\![0, 7]\!]$. En conséquence, presque tous les bits deviennent dépendants des bits de l'opcode.

La propagation des bits défectueux en cas de séquence de décodages valides est garantie par une permutation du masque, qui ne génère aucun surcoût matériel. En effet, il est prévu que les fautes se diffusent à travers l'ensemble du masque si les instructions demeurent valides après plusieurs cycles de décodage. Pour les propriétés de diffusion, la permutation à 32 bits de l'algorithme de chiffrement DES, présentée dans le tableau 4.2, est sélectionnée.

Ainsi, si une faute survient avant l'entrée de l'étape DECOD, elle entraînera des erreurs

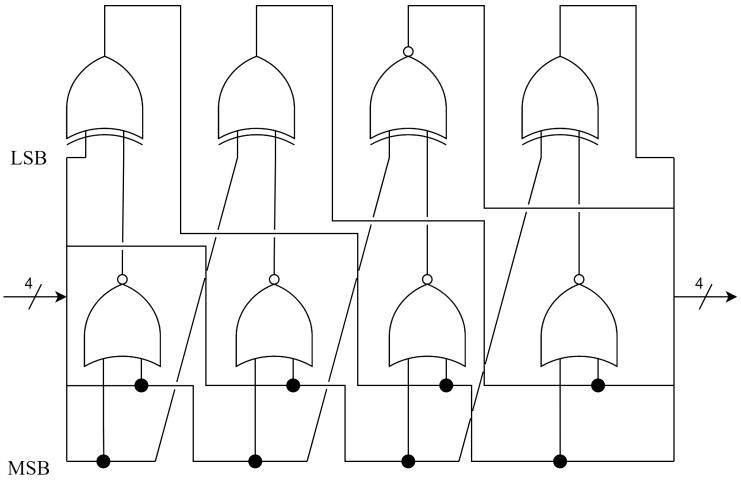

Figure 4.5 : Sbox de l'algorithme Piccolo.

P	16	7	20	21	29	12	28	17
	1	15	23	26	5	18	31	10
	2	8	24	14	32	27	3	9
	19	13	30	6	22	11	4	25

Table 4.2 : Permutation du DES.

multiples lors de la génération du masque, grâce à l'emploi de différentes S-box. La combinaison d'une permutation et de S-box assure la diffusion des fautes à tous les bits de l'instruction en quelques cycles.

4.3.2 Sécurisation de l'étage DECOD

Cette méthode utilise les signaux du post-décodeur pour générer les masques. Les signaux exploités comprennent l'index du registre, l'immédiat, le sélecteur de multiplexeur ainsi que les signaux d'activation. Il est difficile de généraliser, car ces signaux sont spécifiques à chaque architecture, mais dans le cas de RISCY, ils permettent de garantir un décodage correct. Contrairement à la solution utilisant l'instruction précédente, qui ne protège contre les fautes que jusqu'au début de l'étape de décodage, comme illustré à la Figure 4.1, l'utilisation des signaux de sortie du décodeur assure que le décodage s'est effectué sans erreur.

En termes de surcoût matériel ou de taille de la mémoire d'instructions, cette technique est relativement économique. Toutefois, puisque les masques sont générés à partir des signaux de décodage, le chemin critique de cette étape est allongé, ce qui peut entraîner une réduction de la fréquence du processeur. De plus, la compilation devient plus complexe,

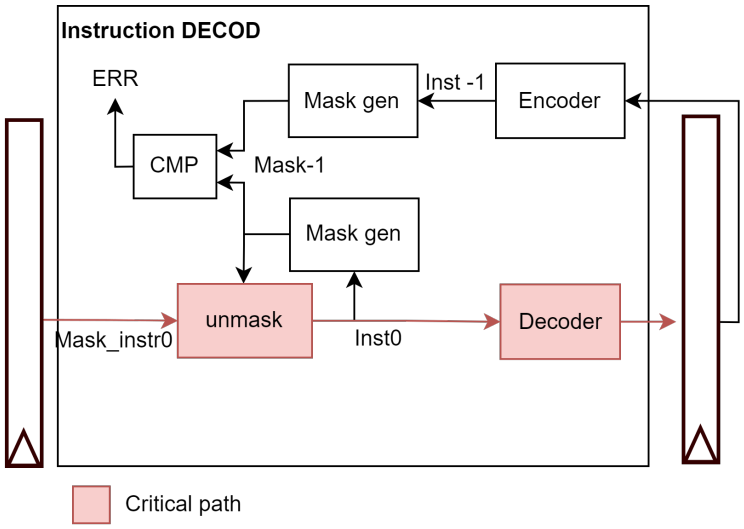

Figure 4.6 : Regénération du masque.

car il faut générer les signaux de décodage de manière logicielle. Un décodeur similaire à celui présent dans l'étape DECOD du processeur doit donc être ajouté au compilateur. Ces deux décodeurs doivent générer les mêmes signaux pour que le décodage soit possible. Le principal inconvénient de cette solution est qu'elle rend le code compilé dépendant non seulement de l'ISA, mais aussi des spécificités de l'implémentation du processeur, en l'occurrence le décodage des instructions.

Regénération du masque

Si l'on utilise l'instruction précédente, il devient impossible de garantir la sécurité du décodage. Il est donc nécessaire de capitaliser sur les différents éléments déjà intégrés. Le plus important d'entre eux est l'enregistrement du masque généré. Il serait donc pertinent de le vérifier au cycle suivant pour assurer la fiabilité du décodage. Les signaux d'entrée de l'étape EXEC peuvent servir d'entrée pour un générateur d'instructions. En effet, il n'y a pas de perte d'information lors du décodage, mais simplement une transformation. Il est donc possible de recoder les signaux de sortie du décodage sous la forme de l'instruction originale. Une fois l'instruction recodée, le masque peut être généré et comparé à celui stocké dans le registre, comme illustré à la Figure 4.6.

Cette solution peut être considérée comme une variante de la duplication de l'étape DECOD, mais elle offre des avantages supplémentaires. Le premier avantage est l'utilisation des signaux sortant du registre entre l'étape DECOD et l'étape EXEC. En exploitant ces si-

gnaux, on sécurise également ce registre. De plus, cette méthode permet d'utiliser une simple duplication qui n'est sensible qu'aux attaques par double injection, considérées comme plus difficiles à mettre en œuvre [Col+21]. En outre, par rapport à une simple duplication, un mécanisme de génération de masque est employé, ce qui entraîne la diffusion des fautes à travers l'ensemble du masque, rendant la détection des fautes encore plus complexe. Le surcoût en termes de surface est comparable à celui du décodage et est effectué en parallèle, ce qui n'entraîne pas d'allongement du chemin critique.

Comparée à la solution utilisant des masques générés à partir des signaux de décodage, cette méthode présente plusieurs avantages. Tout d'abord, la détection n'est plus limitée aux erreurs de décodage, permettant ainsi une identification plus rapide des fautes. Cependant, le surcout matériel est plus important, en raison de l'ajout d'un encodeur et d'un générateur de masque, bien que le chemin critique ne soit pas allongé.

Néanmoins, d'autres problèmes surgissent lorsque l'on crée un masque à partir de l'instruction précédente ou des signaux de décodage. En effet, le code source d'un programme n'est pas parfaitement linéaire : des sauts sont possibles à différents indices du code. Par conséquent, l'instruction précédente dans le fil d'exécution n'est pas nécessairement l'instruction précédente dans le code compilé.

4.3.3 Gestion des masques à la compilation

Ce masquage ne peut être effectué qu'au moment de la compilation, seule étape où la séquence et la signification des instructions en assembleur sont connues. La principale difficulté réside alors dans la gestion des différentes branches possibles du flux d'exécution. Deux cas se présentent. Le premier cas concerne un bloc de base ayant plusieurs blocs successeurs, comme illustré à la Figure 4.7. Il s'agit typiquement du cas des branchements conditionnels.

Pour résoudre ce problème, il est nécessaire de pouvoir générer deux masques distincts à partir d'une même instruction, chacun correspondant à un des chemins possibles. Le second cas se produit lorsque plusieurs blocs de base pointent vers un même bloc de base successeur, comme le montre la Figure 4.8. Dans ce cas, une instruction peut avoir plusieurs instructions précédentes et donc plusieurs masques possibles.

Étant donné que la détermination du bloc de base à l'origine de l'instruction précédente dans le pipeline du processeur est complexe, il n'est pas possible de choisir parmi les

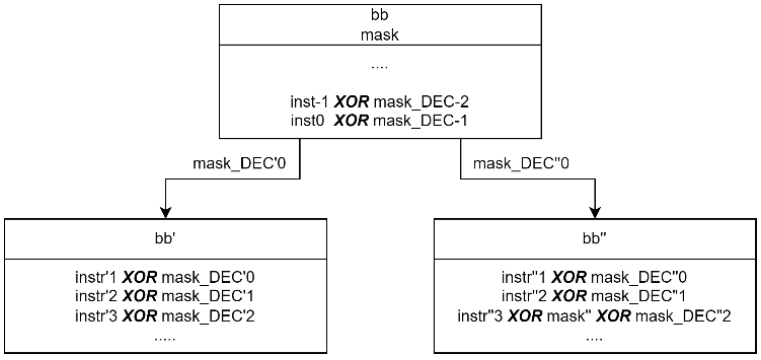

FIGURE 4.7 : Bloc de base avec deux successeurs.

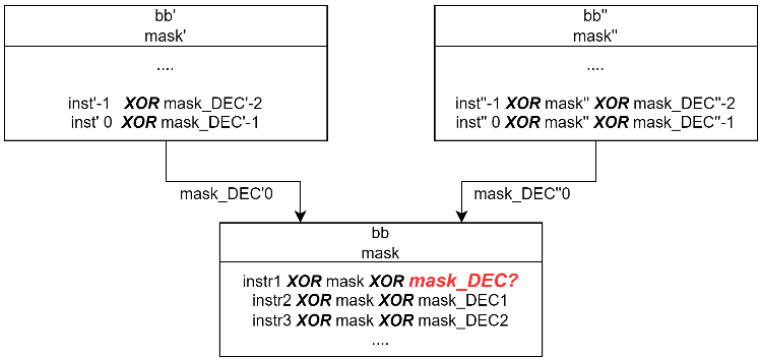

FIGURE 4.8 : Bloc de base avec deux prédécesseurs.

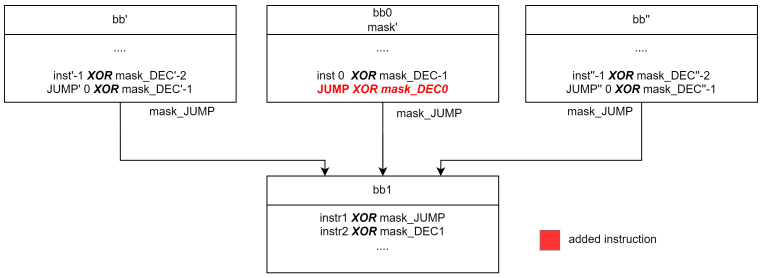

FIGURE 4.9 : Masque généré lors d'un saut.

masques possibles. En général, un masque unique est utilisé pour chaque instruction. Ainsi, si une instruction a plusieurs prédécesseurs, ceux-ci doivent tous générer le même masque, ou cette instruction ne doit pas être soumise à un masquage.

Gestion des instructions de saut

Les instructions de saut, qu'elles soient directes ou indirectes, sont parmi les premières à pouvoir générer des antécédents multiples. En effet, deux instructions de saut peuvent cibler la même adresse de destination. Dans ce cas particulier, le masque associé aux instructions de saut doit être uniforme, indépendamment du caractère direct ou indirect du saut. De plus, ce masque ne doit pas dépendre de l'offset ou d'un registre spécifique, car des instructions de saut avec des offsets différents peuvent converger vers la même instruction cible.

Un autre défi se pose lorsque l'instruction cible d'un saut peut également être atteinte par l'incrémentation standard du compteur de programme du processeur. Dans ce cas, le problème est plus complexe car potentiellement n'importe quelle instruction pourrait précéder la destination du saut. Il est donc impératif que l'instruction immédiatement précédant la cible du saut génère un masque spécifique pour les sauts, que nous pourrions appeler *mask_JUMP*.

Pour résoudre ce problème, la stratégie adoptée consiste à insérer, juste avant chaque destination de saut, une instruction de saut dont la destination est l'instruction suivante. De cette manière, une destination de saut ne peut être atteinte que par des instructions de saut, et le masque généré sera toujours un *mask_JUMP*, comme illustré à la Figure 4.9.

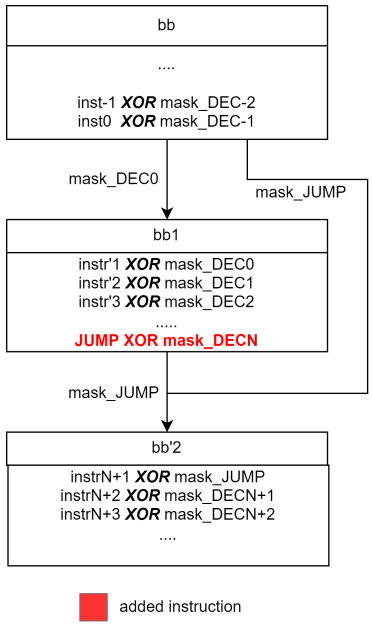

FIGURE 4.10 : Masque lors d'un branchement.

Gestion des branchement

Le cas des instructions de branchement présente des défis similaires à ceux des instructions de saut. En effet, lorsqu'une branche est prise, elle est essentiellement équivalente à un saut. Par conséquent, tous les autres chemins possibles menant à cette instruction doivent générer le même masque. L'insertion d'instructions de branchement supplémentaires avant chaque destination de branchement n'est pas une solution viable, car une branche est par définition conditionnelle. Par conséquent, l'ajout d'une instruction de saut est recommandé. Ainsi, une instruction de saut est insérée avant chaque destination de branchement. De plus, le masque "mask_DEC0", comme illustré à la Figure 4.10, est remplacé par le masque utilisé pour les sauts, "mask_JUMP".

Commutation de contexte et interruptions

Lors d'une commutation de contexte, il est impératif de sauvegarder le masque courant afin de pouvoir démasquer correctement l'instruction à l'adresse de retour. Une étape de sauvegarde du masque est donc ajoutée au début du processus de commutation de contexte, et une étape de restauration du masque est ajoutée à la fin. Cette procédure est analogue à celle de la commutation du compteur de programme (PC).

Dans le cas d'une interruption, le masque qui aurait dû être utilisé dans le cycle suivant est

automatiquement sauvegardé. Pour les architectures basées sur RISC-V, cette sauvegarde est gérée automatiquement au niveau des registres de contrôle et d'état lors de l'interruption.

Gestion des cycles de gel (stall)

Dans certaines situations, un processeur peut être amené à geler son exécution pour diverses raisons, telles que des dépendances de données ou le calcul de la branche à prendre. Pendant ces cycles de gel, le masque à utiliser pour l'instruction suivante est conservé dans un registre interne du processeur.

4.3.4 Modifications au niveau de la compilation

Comme mentionné précédemment, toute instruction accessible par un saut ou une branche doit uniquement être accessible via ces types d'instructions. Une première passe de compilation identifie tous les blocs de base ayant plusieurs antécédents et s'assure que chacun d'eux se termine par une instruction de saut ou de branchement. Si ce n'est pas le cas, une instruction de saut est ajoutée.

Une seconde passe ajoute ensuite les masques à toutes les instructions. Cette passe doit être la dernière étape de la compilation, car elle fixe l'ordre d'exécution du programme. Dès lors, le code ne doit plus être modifié. Cette étape est relativement simple : elle prend l'instruction précédente, génère le masque correspondant et l'applique à l'instruction courante. Le masque est simplement une transformation de l'instruction précédente à l'aide de Sbox et de permutations, des opérations facilement réalisables en logiciel.

Limitations et considérations de sécurité

L'approche présente une limitation en ce qui concerne la sensibilité aux sauts d'instructions. En effet, chaque instruction de destination "JUMP" utilise le même masque. Ainsi, si un saut d'instruction est effectué lors d'une instruction "JUMP" et que ce saut pointe vers une autre destination "JUMP", le masque restera valide. Notre solution ne permet pas de détecter cette faute. Bien que ce scénario soit possible, sa mise en œuvre est complexe, nécessitant de cibler précisément le cycle de saut et de déterminer une nouvelle destination "JUMP" valide.

4.3.5 Implémentation

Afin de concrétiser notre solution, nous nous concentrons sur un processeur RISC-V à quatre étages en ordre séquentiel. Notre mécanisme est intégré à l'étape de décodage du pipeline.

Réalisation matérielle

Un registre nommé `Decod_output` est instancié pour conserver les signaux émanant du décodeur. Ce registre est actualisé à chaque cycle d'horloge, à l'exception des cycles de gel, lesquels sont signalés au processeur via le signal `freeze_sig`. Par ailleurs, le masque employé pour toutes les opérations de saut et de branchement est stocké dans `jump_mask`. La sélection entre `jump_mask` et `DEC_mask` est orchestrée par le `Jump_controller`. Ce dernier utilise les signaux de décodage des instructions ainsi que les signaux en sortie de l'étape d'exécution pour décider si `jump_mask` doit être appliqué. Cette décision englobe les sauts directs et indirects, ainsi que les branches conditionnelles.

L'activation ou la désactivation de la solution est contrôlée par le signal `ACT_DEC_mask`, qui dans notre implémentation est un signal d'entrée du processeur. Ce signal est régulé par un registre externe, accessible via JTAG. Il permet d'exécuter l'instruction entrante soit avec l'application du masque, soit sans. Un problème subsiste quant au premier masque à appliquer, étant donné que la première instruction, par définition, ne possède pas d'instruction précédente. Dans ce cas, c'est `jump_mask` qui est utilisé, nécessitant ainsi de garantir que la première instruction soit correctement masquée par ce dernier.

Pour élargir les options d'activation, un signal de démarrage pourrait être ajouté. Cependant, cela introduirait un vecteur d'attaque supplémentaire et une complexité qui n'apporte pas de bénéfice notable dans notre contexte. Il est important de noter que ce signal ne permet pas de désactiver la contre-mesure, mais s'applique uniquement à l'application du masque ou non.

Bien que les deux solutions soient similaires en termes d'implémentation, la vérification de l'étage `DECOD` avec un pré-décodeur nécessite l'ajout d'un encodeur et d'une vérification du masque, comme illustré à la Figure 4.11. Il est également impératif d'ajouter les signaux manquants pour regénérer l'instruction.

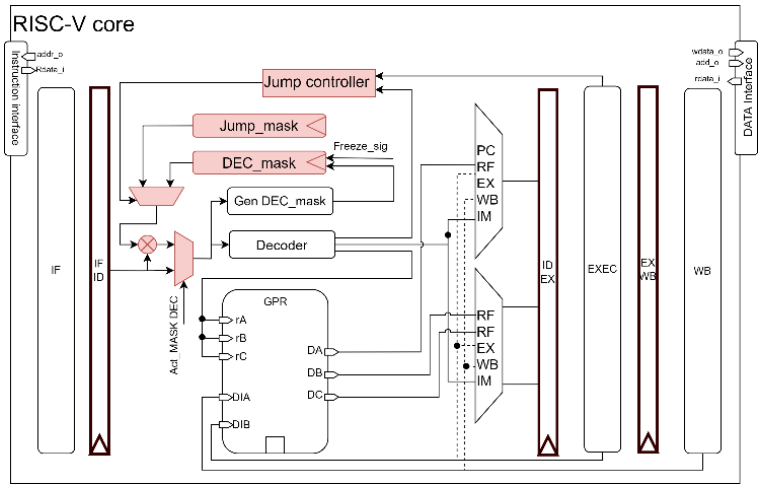

Figure 4.11 : Intégration dans le pipeline du CV32E40P.

Surcoût matériel

Nos architectures RTL, basées sur les processeurs RISCY CV32E40P, ont été synthétisées en utilisant la bibliothèque RVT GF22FDX (GlobalFoundries 22nm FD-SOI) Standard Cells. Le surcoût matériel associé à cette solution est relativement modeste. Les ajouts se limitent au registre de masques, aux opérations de masquage via un XOR, et à la génération des masques. Les résultats de synthèse indiquent que 15 295 GE (équivalents de portes) sont nécessaires avec le post-décodeur et 17 729 GE avec le pré-décodeur. Cela se traduit par une augmentation de la surface de silicium de 3,9% et 20,4% respectivement, et une hausse de la consommation énergétique totale de 4,3% et 19,2%, par rapport au noyau RISCY original qui occupe une surface de 14 727 GEs.

Surcoût logiciel

Les données présentées dans la figure 4.12 illustrent le surcoût en taille de code ainsi qu'en temps d'exécution, induits par les modifications apportées par le compilateur. Ces mesures ont été effectuées en utilisant une simulation au niveau du cycle avec le benchmark Embench1.0. Le surcoût en taille de code est de 1,61%, tandis que celui en temps d'exécution est de 1,12%. Ce surcoût est minime, principalement parce que les modifications apportées par le compilateur et au flux d'instructions sont limitées. En effet, seules des instructions de saut sont occasionnellement insérées avant les destinations de saut.

Notre solution présente plusieurs avantages en matière de sécurité. D'une part, elle est extrêmement légère, nécessitant seulement le stockage de 32 bits de masques et leur appli-

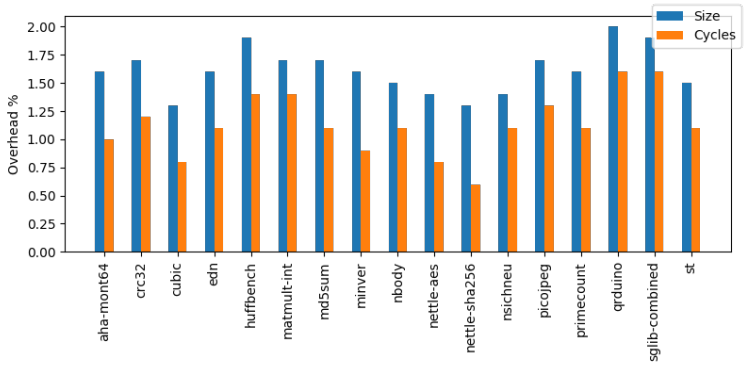

Figure 4.12 : Surcharge en temps d'exécution et en taille de code du benchmark Embench avec cette solution.

cation via une opération XOR à l'instruction entrante du cycle suivant. D'autre part, elle impose un surcoût minime en termes de taille de code et de temps d'exécution. En effet, une instruction de saut est ajoutée avant chaque destination de saut uniquement lorsque cela est requis. De plus, la simplicité de notre approche la rend adaptable aux compilations "just-in-time" et aux codes auto-modifiants, la principale difficulté résidant dans l'insertion des instructions de saut. Enfin, elle offre une protection robuste contre les sauts d'instructions, y compris ceux qui sautent plusieurs instructions, comme souligné par A. Menu et al [Men+20].

L'un des défis majeurs dans l'évaluation des contre-mesures contre l'injection de fautes est l'absence de métriques standardisées pour quantifier leur efficacité. Toutefois, dans notre cas, étant donné que notre modèle d'attaquant est limité au chemin d'instruction, toutes les fautes peuvent être assimilées à des modifications du code machine. Par conséquent, une campagne de tests exhaustifs sur les binaires est possible. Étant donné que les erreurs se propagent de cycle en cycle, nous sommes assurés que toute faute sera détectée dans à un moment donné, sauf en cas de saut valide. Le critère pertinent n'est donc pas le taux de détection, mais plutôt le nombre de cycles avant une erreur de décodage.

Pour évaluer notre solution, nous avons développé un outil logiciel simulant le décodage des instructions RISC-V. Afin de simplifier la création du masque, nous avons opté pour la solution du pré-décodeur qui ne nécessite que l'instruction pour générer le masque. Pour optimiser le temps de calcul, les binaires générés sont exécutés une première fois, puis modifiés exhaustivement lors d'une seconde exécution pour toucher uniquement les instructions effectivement exécutées. Nous avons réalisé 50 tests de vérification fonctionnelle de l'architecture RISC-V pour garantir que tous les types d'instructions sont correctement

pris en compte.

Il convient de noter que nous avons exclu de nos résultats les fautes entraînant un saut valide vers une autre destination de saut. Ces cas, bien que manifestes lors d'une campagne d'injection de fautes exhaustive, demeurent peu probables et ne constituent pas un reflet fidèle d'une campagne d'injection réalisée avec des composants physiques.

Faute simple

Le premier contrôle a pour objectif de valider l'adéquation du masque proposé pour une injection de faute simple. La Figure 4.13 illustre une comparaison entre la solution intégrale pré-décodeur et un masque de 32 bits généré via la fonction de hachage ASCON. Cette comparaison inclut également les deux sous-masques constitutifs : le masque des S-boxs à blocs consécutifs (linear_sbox) et celui à blocs entrelacés (mix_sbox), ainsi que l'instruction sans transformation (identity).

Il est important de noter que la détection au cycle 0 est uniforme pour toutes les variantes de masques, étant donné que le masque n'est pas encore appliqué à ce stade et que notre méthode de test est exhaustive, éliminant ainsi toute variabilité aléatoire. Avec l'instruction non transformée, seulement 35% des fautes subséquentes sont détectées au deuxième cycle. Ce taux s'élève à 45% avec un seul type de S-box, 75% avec le masque complet et atteint 90% avec le hachage généré par ASCON.

Le Tableau 4.3 présente le nombre moyen de cycles nécessaires avant qu'une faute ne provoque une erreur de décodage. Un hachage de l'instruction permet une détection de la faute en deux cycles, tandis que notre solution, qui implique uniquement 16 S-Box matérielles, permet de détecter 97% des fautes en deux cycles. Étant donné que 99% des fautes sont détectées en moins de trois cycles, il est raisonnable de supposer que ces fautes n'ont pas altéré la mémoire de données. Par conséquent, en désactivant le pipeline et le banc de registres, le processeur peut être considéré comme étant dans un état sécurisé.

Fautes multiples

Les attaques par fautes multiples gagnent en popularité et en efficacité, comme le démontre la référence [Col+21], et peuvent éventuellement contourner les mesures de sécurité conçues pour contrer les fautes simples. Néanmoins, notre solution offre une résilience accrue face à une augmentation du nombre de fautes, comme illustré dans la Figure 4.14. Par conséquent, un attaquant n'aurait pas d'intérêt à déployer des fautes plus complexes,

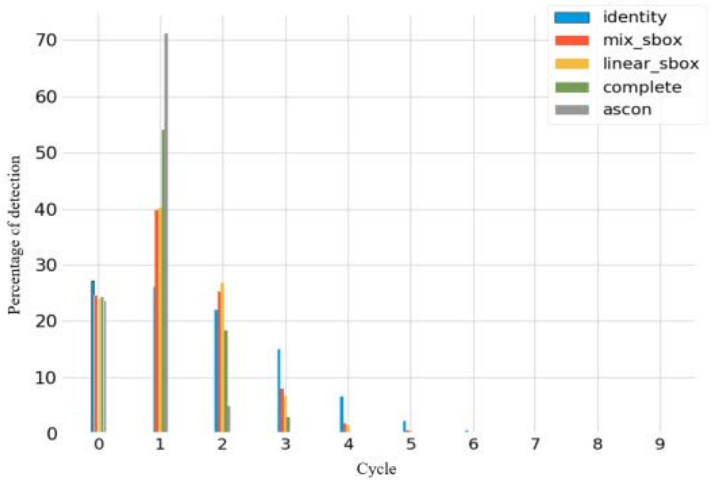

Figure 4.13 : Comparaison des différentes générations de masque pour les fautes simples

	Identity	linear_sbox	Mix_sbox	Complete	ASCON
Average cycle detection	1.52	1.20	1.20	0.99	0.81

Table 4.3 : Comparaison du nombre moyen de détection pour les différentes générations.

à moins de viser une modification très spécifique d'une instruction pour rediriger le flux d'exécution vers une destination de saut valide, ce qui semble actuellement hors de portée des techniques d'attaque existantes.

Saut d'instruction

Outre la modification des instructions, les injections de fautes peuvent également entraîner des sauts d'instructions, qu'ils soient uniques ou multiples. Certaines solutions de sécurité sont sensibles au nombre d'instructions sautées. Cependant, notre approche, illustrée dans la Figure 4.15, permet de détecter ces sauts en moins de deux cycles, indépendamment du nombre d'instructions sautées, que ce soit 1, 2 ou 4. Ainsi, notre solution demeure efficace contre les attaques par fautes sur l'intégralité du chemin d'exécution des instructions. Plus une attaque est complexe, combinant à la fois des fautes et des sauts, plus notre solution s'avère efficace pour la contrer.

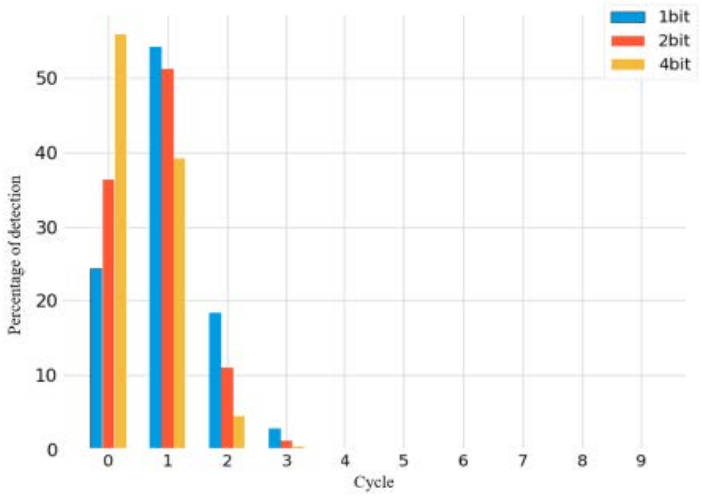

FIGURE 4.14 : Comparaison des différentes générations de masque pour les fautes multiples

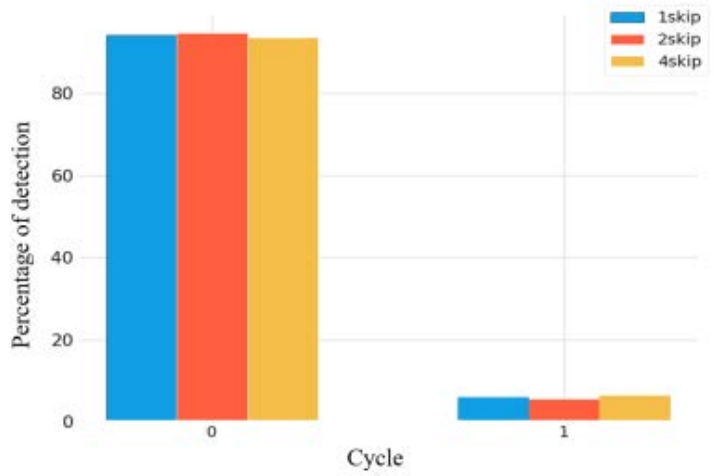

FIGURE 4.15 : Comparaison des différentes générations de masque pour les sauts d'instruction

4.4 Duplication/Parité Croisée sur les Signaux de Contrôle

Les registres et signaux de contrôle jouent un rôle central dans la gestion et l'état du processeur, ce qui en fait des cibles potentiellement intéressantes pour les injections de fautes. Néanmoins, ces composants sont généralement moins sensibles aux attaques par canaux auxiliaires pour plusieurs raisons. Premièrement, leur nature fonctionnelle et leur emplacement au sein de l'architecture du processeur les rendent difficiles à cibler avec précision. Deuxièmement, ils ne contiennent généralement pas de données sensibles, ce qui réduit leur valeur pour un attaquant cherchant à extraire des informations confidentielles.

Cependant, leur rôle dans la coordination des différentes unités fonctionnelles du processeur et dans la gestion des états de la machine les rend vulnérables aux attaques visant à perturber le fonctionnement normal du système. Une injection de faute réussie sur ces composants pourrait, par exemple, entraîner une séquence d'opérations incorrecte, compromettant ainsi l'intégrité du système dans son ensemble.

Pour pallier ces vulnérabilités, diverses techniques de protection peuvent être envisagées, telles que la duplication des registres et la parité croisée sur les signaux de contrôle. Ces méthodes visent à augmenter la redondance et à permettre une vérification en temps réel de l'intégrité des signaux, minimisant ainsi les risques associés aux injections de fautes.

4.4.1 Description Détaillée

L'un des attributs essentiels d'un code correcteur d'erreurs efficace contre les attaques par injections de faute est sa forte localité spatiale dans le domaine des fautes. En d'autres termes, il est crucial que deux fautes spatialement proches ne puissent pas être simultanément valides. Pour maximiser cette distance spatiale, nous avons recours à la technique de la parité.

La parité est particulièrement efficace pour détecter les fautes impaires dans un mot binaire. En entrelaçant les bits de parité, il devient pratiquement impossible d'injecter deux fautes consécutives sans détection. Prenons l'exemple de la vérification de parité longitudinale (Longitudinal Parity Check, LPC), illustrée dans la Figure 4.16. Tout ensemble de fautes consécutives qui ne correspond pas exactement au double de la longueur du LPC sera détecté.

L'avantage de la parité est sa facilité de combinaison avec d'autres codes détecteurs. Par exemple, en combinant la vérification de parité longitudinale (LPC) avec la vérification de

parité horizontale (Horizontal Parity Check, HPC), on obtient un code détecteur capable de détecter toute faute ne correspondant pas à un schéma prédéfini.

Pour renforcer davantage la robustesse du système, il est également possible d'intégrer des vérifications de parité diagonales. Cependant, cette amélioration a un coût, tant en termes de mémoire que de logique combinatoire. Pour calculer une parité, il faut n opérations XOR, et pour stocker les bits de parité, $p\sqrt{n}$ bascules D (flip-flops) sont nécessaires, où n est le nombre de bits du mot à protéger.

En somme, l'utilisation de techniques de parité, seules ou en combinaison, offre une méthode efficace et flexible pour améliorer la résilience du système contre les attaques par injections de faute, tout en tenant compte des contraintes de ressources.

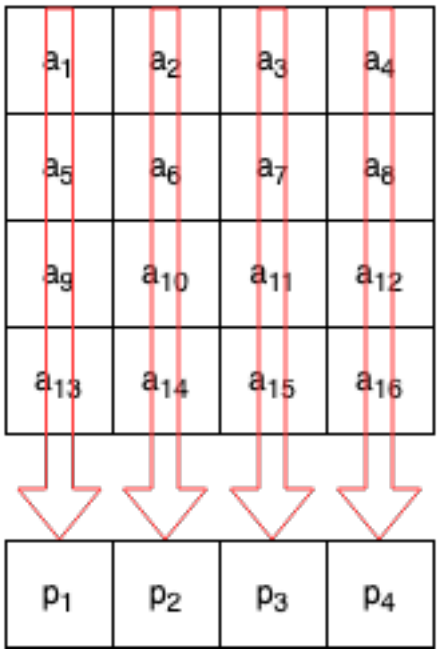

Figure 4.16 : Parité longitudinale

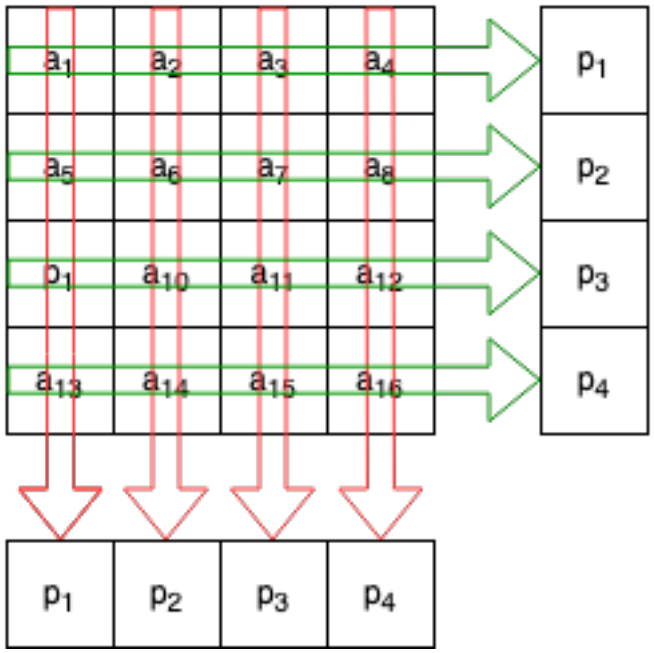

Figure 4.17 : Parité croisée

4.4.2 Avantage

Le premier avantage notable de cette solution réside dans son coût d'implémentation relativement faible. En effet, l'utilisation de la parité nécessite uniquement des portes XOR, ce qui en fait une option économique en termes de ressources matérielles. Cette légèreté est particulièrement avantageuse car elle permet d'effectuer des vérifications à chaque entrée et sortie des différents étages du système, ainsi qu'au niveau des registres d'état.

De plus, en cas de besoin accru de sûreté, il est possible d'intégrer une correction d'erreur monobit. Grâce à l'entrelacement des deux types de parité (horizontale et longitudinale), il est possible de localiser précisément les coordonnées du bit fautif dans la matrice de parité.

L'utilisation de la parité croisée induit également une dépendance spatiale entre les fautes. Pour induire une faute valide sur un bit spécifique, il faudrait injecter au minimum trois fautes à des emplacements spécifiques. Cette caractéristique contraste avec les codes détecteurs d'erreurs traditionnels, qui ne peuvent que détecter une distance minimale entre les fautes. Dans ces derniers, une source unique de faute peut plus facilement induire plusieurs erreurs consécutives. En revanche, avec la parité croisée, des schémas d'injection spécifiques assortis de fortes contraintes spatiales doivent être respectés pour induire une faute valide.

4.5 Conclusion

Ce chapitre a exploré deux méthodes innovantes pour renforcer la sécurité du chemin des instructions contre les attaques par défaut et par canal latéral. La première méthode s'appuie sur les instructions précédentes pour générer des masques de protection, tandis que la seconde utilise les signaux de décodage de l'instruction antérieure. Ces stratégies présentent l'avantage considérable de minimiser l'overhead, tant au niveau matériel que logiciel. Plus précisément, l'utilisation des signaux de décodage entraîne un surcoût matériel de seulement 3,25% et assure une détection des fautes en un cycle en moyenne. Cependant, cette approche rend la compilation sensible aux contraintes micro-architecturales. À l'opposé, en se basant uniquement sur les informations du pré-décodeur, nous parvenons à sécuriser le chemin des instructions avec un doublement de l'étage de décodage, tout en rendant le code source dépendant uniquement du jeu d'instructions. L'impact sur les ressources logicielles est également modeste, avec un overhead de 1,61% en taille de code et de 1,12% en temps d'exécution, grâce à des ajustements minimes au niveau du compilateur.

En outre, il est important de mettre en lumière l'efficacité de l'incorporation de techniques de parité, notamment la parité croisée, dans notre architecture de sécurité. Ces techniques ajoutent une couche supplémentaire de protection en induisant une dépendance spatiale entre les fautes, rendant ainsi les attaques plus difficiles à exécuter. La parité croisée, en particulier, nécessite des schémas d'injection de fautes très spécifiques et contraints spatialement pour être efficaces, ce qui augmente considérablement la robustesse du système.

En somme, les solutions proposées dans ce chapitre offrent une protection complète contre les attaques sur le chemin des instructions. Elles complètent ainsi efficacement les mesures de sécurité dédiées au flux de contrôle et au chemin des données. Cette approche permet une sécurisation globale du système, optimisant l'équilibre entre robustesse et efficacité des ressources.

5

Sécurisation et intégration combinées

Après avoir élaboré des contre-mesures pour le chemin de données et le chemin de contrôle, nous disposons d'une protection a priori exhaustive du pipeline du processeur contre les attaques par injection de fautes et par canaux auxiliaires. Toutefois, au lieu de considérer ces deux chemins de manière isolée, il est pertinent d'examiner l'architecture globale du processeur afin de déterminer si des contre-mesures supplémentaires peuvent être instaurées. Dans ce chapitre, nous introduisons une contre-mesure qui exploite les principes fondamentaux des processeurs, à savoir l'exécution d'instructions, dans le but de complexifier toute forme d'attaques physiques. Ensuite, nous discuterons de l'intégration de ces contre-mesures au sein d'un pipeline d'un processeur RISC-V 32 bits. Nous mettrons en lumière les diverses interactions entre les contre-mesures qui augmentent la sécurité, ainsi que leur adaptabilité à cette architecture spécifique.

5.1 Désynchronisation

L'introduction de délais aléatoires dans l'exécution d'un algorithme cryptographique constitue une contre-mesure à la fois simple et efficace face aux attaques par injection de fautes et par canaux auxiliaires. Selon nos observations, ces délais aléatoires sont fréquemment employés pour sécuriser les implémentations cryptographiques dans les systèmes embarqués, notamment les cartes à puce, en raison de leur faible coût additionnel. Cette technique s'inscrit dans une catégorie de contre-mesures d'obscurcissement qui introduisent un bruit additionnel dans le système. L'association de diverses méthodes d'obscurcissement et de masquage est couramment utilisée pour accroître la complexité des attaques potentielles.

La majorité des attaques par canaux auxiliaires et par injections de fautes nécessitent que l'attaquant connaisse avec précision le moment où les opérations cibles sont exécutées. Cette connaissance permet de synchroniser plusieurs traces lors d'événements critiques, comme c'est le cas dans l'analyse de puissance différentielle, ou d'injecter une perturbation calculée au moment opportun, comme dans les attaques par injections de fautes. L'introduction de délais aléatoires perturbe cette synchronisation, complexifiant ainsi l'attaque. Diverses techniques de randomisation temporelle ont été proposées dans la littérature, par exemple [CK09 ; CCD00]. On peut généralement distinguer les méthodes logicielles, telles que celles basées sur les retards aléatoires par interruption (RDI) qui consiste à utiliser des interruptions systèmes pour insérer des instructions factices mais ces instructions factices peuvent aussi directement etre injecter par du materiel specifique comme softRJJD ou ERIST. D'autre méthodes matérielles existent comme celles qui augmentent le jitter de l'horloge. Plus les contre-mesures sont intégrées au niveau matériel, plus les stratégies pour les contourner se concentrent sur le traitement du signal [Gui+11 ; WWB11]. Il est à noter que de nombreuses évaluations de l'efficacité des contre-mesures, par exemple [Man04], prétraitent les traces de fuite en les intégrant. Des methodes specifique a la decterions d'insertions d'instructions factices ont ete devellope en ce basant sur la reconnaissance de motifs peuvent éliminer les ajouts d'instructions [Dur+12]. Influencées par cette méthode d'évaluation, des recherches visent à maximiser la variabilité des injections de d'instructions factices afin d'améliorer la distribution statistique de l'échantillon aléatoire des retards, dans le but de produire des traces aussi bruyantes que possible. En somme, pour que les injections de code factice soient efficaces, il est nécessaire d'augmenter la distribution statistique de l'échantillon aléatoire des retards et de veiller à ce que ces injections

ne suivent pas un motif reconnaissable.

La plupart des attaques par canaux auxiliaires et par injections de fautes requièrent une connaissance précise par l'attaquant du moment où les opérations cibles sont exécutées. Cette connaissance facilite la synchronisation de multiples traces lors d'événements critiques, comme c'est le cas dans l'analyse de puissance différentielle, ou permet l'injection d'une perturbation calculée au moment opportun, comme dans les attaques par injections de fautes. L'ajout de délais aléatoires brouille cette synchronisation, rendant ainsi l'attaque plus complexe. Diverses techniques de randomisation temporelle ont été suggérées dans la littérature, par exemple [CK09 ; CCD00]. On peut typiquement distinguer les méthodes logicielles, telles que celles basées sur les retards aléatoires par interruption (RDI) qui consiste à utiliser des interruptions systèmes pour insérer des instructions fictives, mais ces instructions fictives peuvent aussi être injectées directement par du matériel spécifique comme softRJJD ou ERIST. D'autres méthodes matérielles existent, comme celles augmentant le jitter de l'horloge. Plus les contre-mesures modifient les paramètres physiques du matériel, plus les stratégies pour les contourner se focalisent sur le traitement du signal [Gui+11 ; WWB11]. Il est à noter que de nombreuses évaluations de l'efficacité des contremesures, par exemple [Man04], prétraitent les traces de fuite en les intégrant. Des méthodes spécifiques à la détection d'insertions d'instructions fictives ont été développées, se basant sur la reconnaissance de motifs, et peuvent éliminer les ajouts d'instructions [Dur+12]. Influencées par cette méthode d'évaluation, des recherches visent à maximiser la variabilité des injections d'instructions fictives afin d'améliorer la distribution statistique de l'échantillon aléatoire des retards, dans le but de produire des traces aussi bruyantes que possible. En somme, pour que les injections de d'instructions fictives soient efficaces, il est nécessaire d'augmenter la distribution statistique de l'échantillon aléatoire des retards et de veiller à ce que ces injections ne suivent pas un motif reconnaissable

Dans [Dur+12], les auteurs établissent des critères pour une insertion efficace d'instructions factices, qui devraient :

- présenter des délais sans motif régulier,
- être imprévisibles,
- ressembler aux instructions environnantes.

Les motifs réguliers sont faciles à éliminer dans les méthodes logicielles, car ces instructions nécessitent souvent des prologues pour appeler les routines d'insertion factice. De

plus, dans de nombreuses applications, l'augmentation de la taille du code n'est pas une option viable. Seule l'insertion matérielle en temps réel d'instructions factices permet d'éviter cette contrainte. Cependant, les solutions matérielles actuelles, telles que [He+16], [ARP12], répondent aux deux premiers critères en insérant aléatoirement des instructions sans motif, mais elles posent le problème de la distinction entre les instructions factices et les instructions réelles, ainsi que de la diversité et de la cohérence des instructions insérées.

5.1.1 Présentation de la Contre-Mesure

La contre-mesure que nous proposons est un dispositif matériel capable d'insérer des instructions factices à des intervalles aléatoires durant l'exécution du programme. L'objectif est de concrétiser le concept de cycle factice, couramment utilisé au stade de la compilation dans la plupart des travaux antérieurs. Pour simplifier la description de la mise en œuvre, nous nous situons toujours dans le contexte de la micro-architecture CV32E40P. Il est à noter que le choix de l'architecture du jeu d'instructions et de l'architecture du processeur n'affecte pas la validité de la solution proposée.

L'architecture de la contre-mesure suggérée, illustrée à la Figure 5.1, se divise en deux composantes principales : la génération de l'instruction et son injection dans le pipeline du processeur. La première composante comprend un registre "dummy opcode" et un bloc "Generate random instr" pour la création de l'instruction factice. La seconde composante gère l'insertion de cette instruction dans le pipeline du processeur. Cette insertion est orchestrée par un multiplexeur, contrôlé par un diviseur de fréquence programmable, nommé "Variable div clock", et un bit aléatoire généré par un générateur de nombres aléatoires ("RNG"). Un signal "flag dummy" est également propagé pour informer les étapes subséquentes du pipeline qu'une instruction factice est en cours d'exécution.

Génération de l'instruction factice

Le défi inhérent à la génération d'instructions factices réside dans la capacité à produire des instructions aussi similaires que possible aux instructions légitimes. La méthode la plus couramment employée consiste à générer ces instructions fictives à partir d'opcodes d'instructions arithmétiques prédéfinis.

Notre approche permet de les créer en se basant sur des instructions déjà traitées, ce qui autorise l'utilisation de presque toute l'ISA comme source d'instructions factices. Seuls les éléments indispensables à la détermination du type d'instruction doivent être conservés.

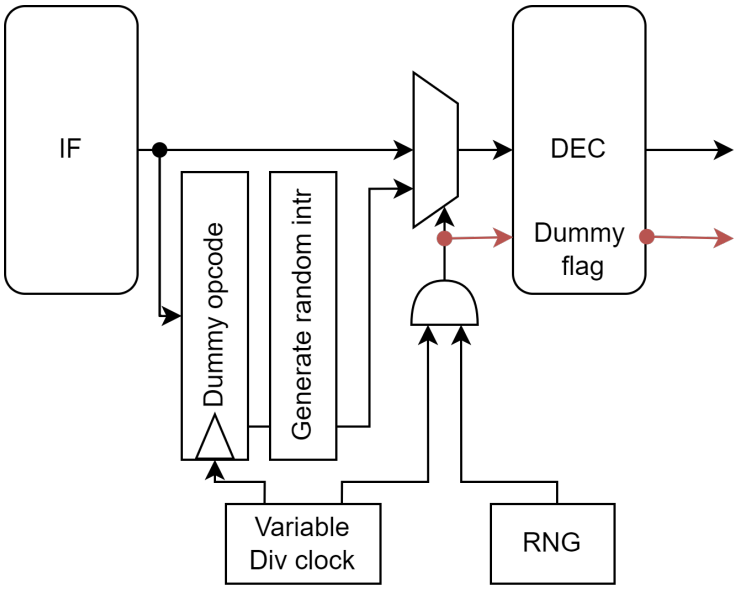

Figure 5.1 : Architecture de l'insertion d'instruction factice.

En examinant les différents types d'instructions dans la Figure 5.2, il est évident que seuls 17 bits des champs opcode, funct3 et funct7 sont nécessaires pour le décodage de l'instruction. Une attention particulière doit être accordée aux instructions qui requièrent un niveau de privilège spécifique. Dans l'architecture RISC-V, ces instructions sont regroupées sous le SYSTEM opcode et sont omises dans notre solution afin d'éviter toute incohérence de privilège lors de l'exécution d'instructions factices.

Si seulement ces 17 bits sont conservés et que les autres bits peuvent être choisis aléatoirement, le décodage reste valide. Toutefois, générer 15 bits aléatoires requerrait une implémentation matériellement onéreuse si l'on souhaite un aléa de niveau cryptographique. Il est donc plus judicieux de choisir ces 15 bits à partir d'un état interne du processeur, tel que le registre de contrôle/état (CSR), les registres généraux, ou le compteur de programme (PC). Le choix de cette source est peu contraignant ; elle doit simplement varier fréquemment et ne pas contenir de données sensibles. Cette méthode présente également un faible risque en termes de sécurité, car les données manipulées sont utilisées uniquement pour la sélection de registres ou d'immédiats. Elle offre des entrées différentes pour les instructions à chaque exécution et ajoute de la variabilité en randomisant le poids de Hamming de l'instruction factice ainsi que la distance de Hamming par rapport aux instructions légitimes adjacentes.

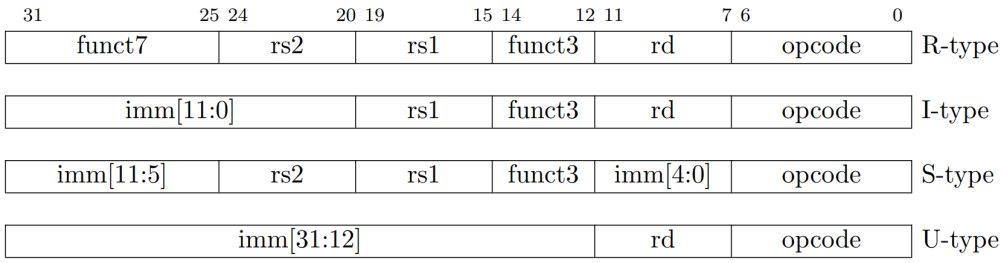

Figure 5.2 : Types d'instructions RISC-V.

Insertion de l'instruction factice

L'insertion des instructions factices doit être imprévisible, ce qui nécessite l'introduction d'aléa dans le processus de décision d'insertion. Dans notre solution, un diviseur de fréquence variable génère un signal d'horloge à la fréquence $F \times 2$, où F est la fréquence cible pour l'insertion des instructions factices. F peut être défini par un registre de configuration, tel qu'un CSR. Pour des raisons pratiques, notre implémentation utilise un registre externe de 32 bits, modifiable via JTAG. L'insertion d'une instruction est conditionnée par un multiplexeur, commandé par la sortie d'une porte ET dont les entrées sont la sortie d'un générateur de nombres aléatoires et la sortie du diviseur de fréquence. La probabilité que le signal de sortie du diviseur de fréquence passe à travers la porte ET est de $\frac{1}{2}$, ce qui donne une fréquence moyenne d'insertion d'instruction factice de F. À chaque période du signal du diviseur de fréquence, le système récupère une instruction et la stocke dans le registre "dummy opcode". La variabilité des instructions insérées est ainsi accrue, car les opcodes stockés ne sont utilisés qu'une seule fois.

L'insertion des instructions factices se réalise à l'étape DECOD du pipeline. À ce moment-là, l'étape IF est gelée, et l'exécution reprend au cycle suivant de manière transparente pour le processeur. Ainsi, le flux normal d'exécution n'est pas perturbé et les mécanismes d'interruption demeurent fonctionnels.

Il est crucial de s'assurer que l'instruction factice et les signaux générés pendant son exécution ne perturbent pas le flux normal d'exécution du programme. À cet effet, un signal "dummy flag" est propagé pour informer le processeur qu'une instruction factice est en cours d'exécution.

5.1.2 Modification de l'architecture

Il est difficile de définir toutes les modifications car elles dépendent fortement de l'architecture et de la mise en œuvre du processeur. Cependant, certaines considérations générales peuvent être soulevées. Tout d'abord, lorsque le processeur exécute une instruction fictive, aucune écriture en mémoire ou dans les registres légitimes n'est autorisée pour des raisons évidentes d'intégrité de la mémoire. En outre, il faut éviter que le processeur modifie les registres de contrôle, les drapeaux ou le PC ; par exemple, les instructions de saut et de branchement. Il est également nécessaire d'éviter que les résultats soient contournés. Comme proposé ci-dessous, il pourrait être intéressant de modifier l'architecture pour que ces cycles fictifs ressemblent davantage à des instructions légitimes.

Écriture dans les registres et la mémoire

L'écriture dans le banc de registres constitue un point discriminant pouvant permettre d'identifier des instructions factices. Aucun changement n'est nécessaire pour les lectures car seules les écritures influencent l'exécution légitime. Pour les écritures, on se limite aux registres non utilisés par le flux normal d'exécution. Toutefois, tous les registres pouvant potentiellement être utilisés, il est impératif de prévoir deux registres de destination supplémentaires, étant donné les deux voies d'écritures de registre dans le processeur CV32E40P. Cependant, écrire systématiquement dans les mêmes registres est identifiable. Il est envisageable de recourir à un banc de registres dynamique, comme on peut en trouver dans [MMS01]. Cette idée a émergé naturellement durant le processus de réflexion. C'est à la suite d'une recherche d'antécédents en vue de breveter cette invention que nous avons découvert cet article.

La banque de registres dynamique, illustrée dans la Figure 5.3, est rendue possible en séparant les indices de registres ciblés par l'ISA des registres physiques. L'indice fait désormais référence à une table de recherche pointant vers un emplacement de registre physique. Une table de validité des registres physiques doit être conservée afin de déterminer quel registre est utilisé.

La table de validité consiste à ajouter un bit de validité à chaque registre dans la banque. Lorsqu'un registre physique est écrit, il est considéré comme valide. Il existe deux façons d'invalider un registre, la première est lors d'un retour de fonction. En effet, lorsqu'une instruction "RET" est exécutée, une partie des registres est sauvegardée et l'autre non. Ces registres non sauvegardés peuvent être considérés comme invalides. L'autre façon de désac-

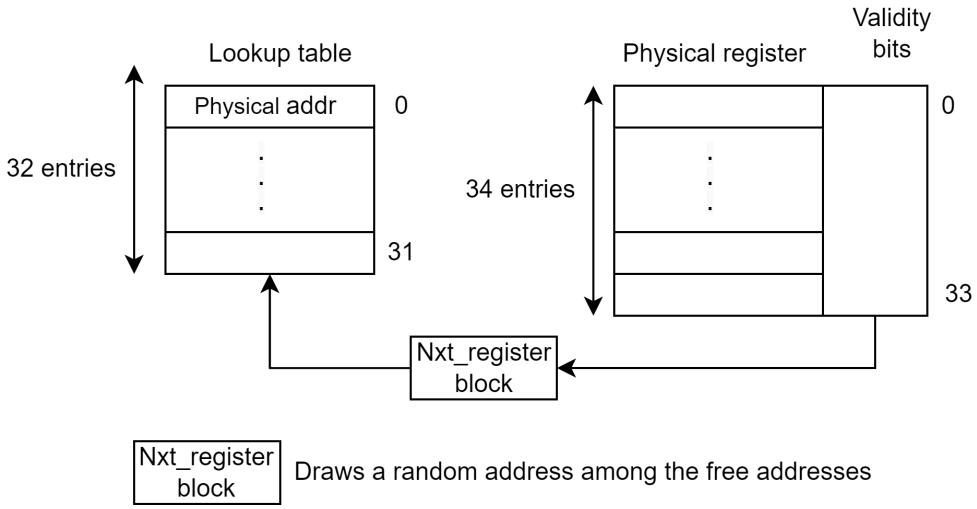

Figure 5.3 : Banc de registre dynamique.

tiver un registre physique est de maintenir la nature dynamique de la banque de registres. Lors de l'écriture dans la banque de registres, le registre physique associé au registre ISA à écrire est d'abord invalidé, puis un nouveau registre physique est choisi parmi les registres libres. Ainsi, à chaque écriture dans un registre ISA, celui-ci est écrit dans un registre physique différent. La figure 5.4 illustre une écriture dans un registre ISA déjà assigné à un registre physique.

Le bloc "Nxt_register" génère un indice de registre aléatoire parmi les registres invalides. La structure arborescente illustrée en figure 5.5 facilite la sélection du registre invalide dans lequel l'instruction fictive va écrire, simplifiant ainsi la sélection d'un registre aléatoire parmi les registres libres. Cependant, il doit toujours y avoir au moins un registre libre. C'est le nombre aléatoire R qui détermine la sortie Out. Si l'un des registres n'est pas invalide, la sortie est l'autre registre. Le cas où deux registres sont invalides n'est pas problématique, car il y a toujours un registre invalide, et on restera donc dans les branches de l'arbre où au moins un registre est valide. Ce composant est dénommé Module de Sélection Aléatoire (RS Module). Pour notre structure arborescente, il est essentiel de généraliser ce composant en sélectionnant la bonne branche à chaque niveau de l'arbre. Pour ce faire, on dispose $N/2$ modules RS, avec N représentant le nombre d'entrées. Le choix entre les $N/2$ modules est guidé par les modules RS des niveaux supérieurs dans l'arbre. L'indice du registre écrit par l'instruction est obtenu par la concaténation des sorties des différents

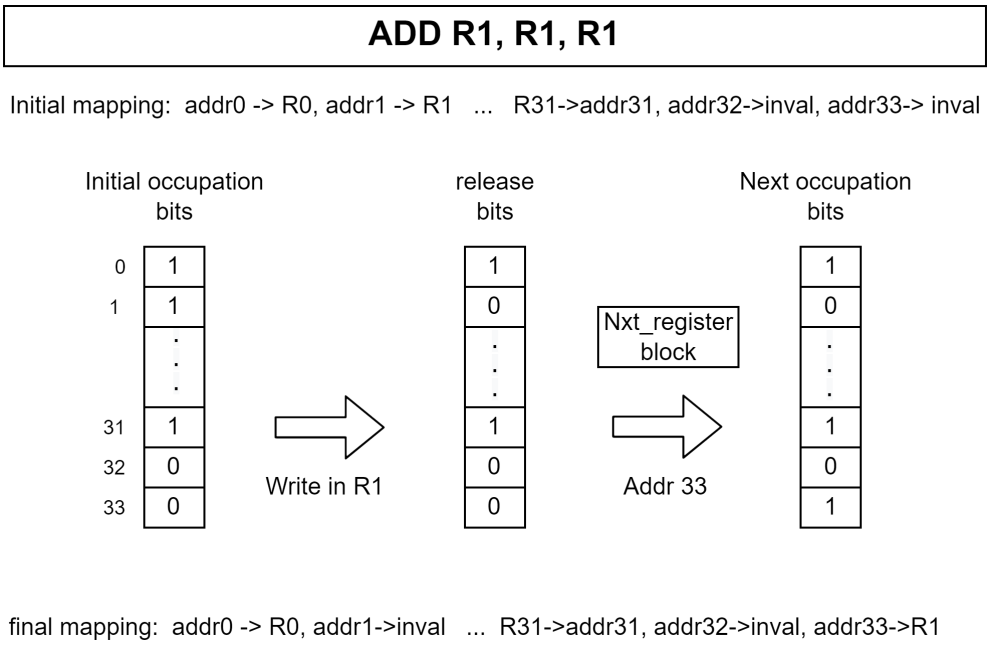

Figure 5.4 : Mise a jour de la validité des registres.

modules RS. Lorsque des instructions factices écrivent dans des registres, le registre écrit ne doit pas être validé. De plus, il n'est pas nécessaire d'effectuer l'étape d'invalidation mentionnée précédemment.

Dans notre cas nous avons jusqu'à deux écriture dans les registres par cycle grâce à cette structure en arbre nous pouvons prendre l'inverse des bits sélection pour trouver un autre registre libre. Cependant pour gérer le cas où le banc de registre est complet, il nous faut en permanence 2 registres vides. Le deuxième avantage est un plus grand choix lors de l'écriture car quand le banc de registre est plein nous avons 2 choix parmi trois pour écrire dans un nouveau registre. Alors quand sans ces registres supplémentaires il n'y a plus de variabilité. Par conséquent, avec le registre dynamique, la seule différence discernable, du point de vue de l'attaquant, pour la manipulation de registres factices réside dans la gestion des bits de validité. La gestion des instructions Load/Store est une préoccupation en raison de la difficulté à déterminer les espaces mémoire libres. Nous résolvons ce problème en attribuant une adresse unique à la lecture et à l'écriture de la mémoire.

Sauts et branchements

Les instructions de branchement et de saut sont couramment utilisées dans les programmes, et il est donc impératif de trouver une manière de les exécuter tout en restant

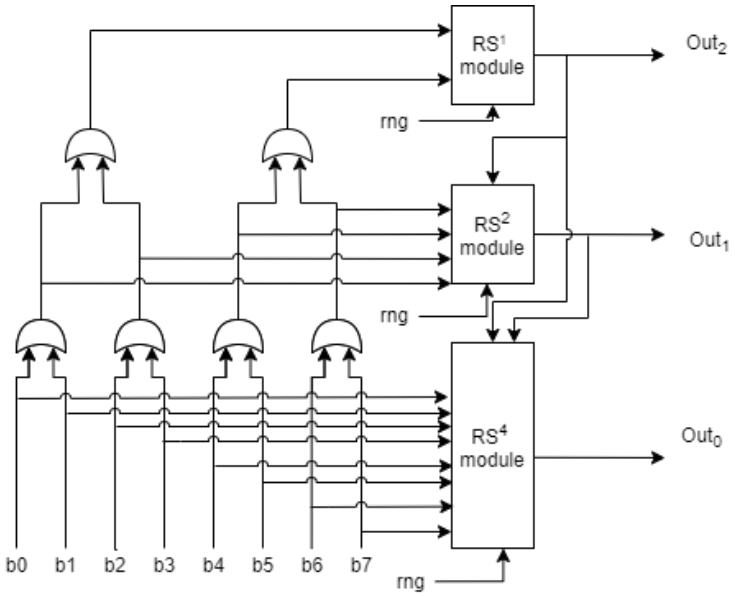

Figure 5.5 : Sélection du registre libre.

aussi fidèle que possible au comportement original du code. Pour les instructions de saut, nous utilisons la même opération que celle spécifiée, mais au lieu de sauter à l'adresse indiquée dans l'opérande ou le registre, l'exécution se poursuit simplement à l'instruction suivante dans la séquence.

La même stratégie ne peut pas être appliquée aux instructions de branchement. En effet, si un branchement est effectué, le processeur doit annuler les instructions qui ont été exécutées avant que le résultat du branchement ne soit connu. Cela signifierait que ces instructions seraient réexécutées, posant ainsi un problème non seulement en termes de détection des instructions factices, mais aussi en termes de potentielles fuites d'informations via des canaux secondaires. Par conséquent, nous avons pris la décision de ne pas inclure les instructions de branchement parmi les instructions factices dans notre implémentation.

5.1.3 Comparaison

En comparaison avec d'autres solutions matérielles, notre approche offre une flexibilité accrue en permettant l'insertion de toute une gamme d'instructions, y compris les opérations logiques et arithmétiques, les sauts et les accès mémoire. D'autres méthodes se limitent souvent à l'insertion de quelques instructions arithmétiques, ce qui restreint leur applicabilité principalement aux primitives cryptographiques. En plus de la désynchronisation temporelle, notre solution intègre un banc de registres dynamique, rendant plus

complexes les attaques par canaux auxiliaires et les attaques par injection de fautes sur la banque de registres. Notre architecture a été implémentée en RTL sur la base du processeur RISC-V CV32E40P et synthétisée en utilisant la bibliothèque de cellules standard GF22FDX (GlobalFoundries 22nm FD-SOI) RVT. Les résultats indiquent un coût en termes de portes équivalentes (GE) de 15395 avec shadow register et de 17540 avec registre dynamique. Cela représente une augmentation de la surface de 4,5 % et 19,1 % respectivement, ainsi qu'une augmentation de la consommation totale d'énergie de 4,27 % et 10,19 % par rapport au cœur RISC-V original, qui a une surface de 14 727 GE. À titre de comparaison, le surcout est de 6,33 % pour softRIJJD et de 8 % pour ERIST lorsqu'implémentés sur une architecture ARM7 de taille initiale de 30081 GE, synthétisée via le compilateur de conception Synopsys sur la base d'une bibliothèque de cellules standard UMC 0,18 μm [He+16].

Concernant le surcout en termes de performances, celle-ci dépend du niveau de sécurité et de l'augmentation du temps d'exécution souhaité. Dans d'autres solutions, ce rapport varie généralement entre 15 % et 25 % [He+16], [ARP12].

Un autre aspect qui a reçu peu d'attention est la capacité à distinguer les instructions factices des instructions authentiques au niveau matériel. Les méthodes classiques de resynchronisation, telles que la corrélation croisée ou les modèles de Markov cachés, ne semblent pas capables de les différencier efficacement [He+16]. Cependant, il est envisageable que des réseaux neuronaux puissent être entraînés pour éliminer ces instructions en apprenant les différences induites, à l'instar des contre-mesures basées sur le jitter d'horloge [Cag18]. Bien que nous proposions des améliorations pour rendre cette distinction plus difficile à réaliser, des évaluations supplémentaires sont nécessaires pour déterminer si cette différence est suffisamment significative pour les divers types d'instructions afin d'être exploitée pour la resynchronisation des traces.

5.2 Implémentions dans le VT2/résultats

Dans cette section, nous nous concentrerons sur un aspect spécifique de la sécurité des systèmes embarqués : l'intégration du tag homomorphique dans un processeur RISC-V 32 bits. Alors que les autres contre-mesures de sécurité ont déjà été discutées en détail dans les chapitres qui leur sont consacrés. Nous aborderons les défis et les solutions associés à cette intégration, en mettant un accent particulier sur les mécanismes additionnels qui ont été développés pour évaluer la robustesse du système dans son ensemble. Nous examinerons également les modifications spécifiques qui ont été apportées au cœur du processeur pour

accommoder cette fonctionnalité de sécurité avancée. Enfin, une analyse du surcoût global, tant en termes de ressources matérielles que de performances, sera présentée pour évaluer l'efficacité et la viabilité de notre solution intégrée.

5.2.1 Véhicule de test

Le véhicule de test est composé de deux processeurs, comme illustré en Figure 5.6. Le premier est un processeur standard CV32E40P, tandis que le second est une variante modifiée intégrant plusieurs contre-mesures de sécurité, à savoir :

- Les registres dynamiques
- SECDEC (Secure Error Detection and Correction)
- Les instructions factices
- Les tags d'intégrité

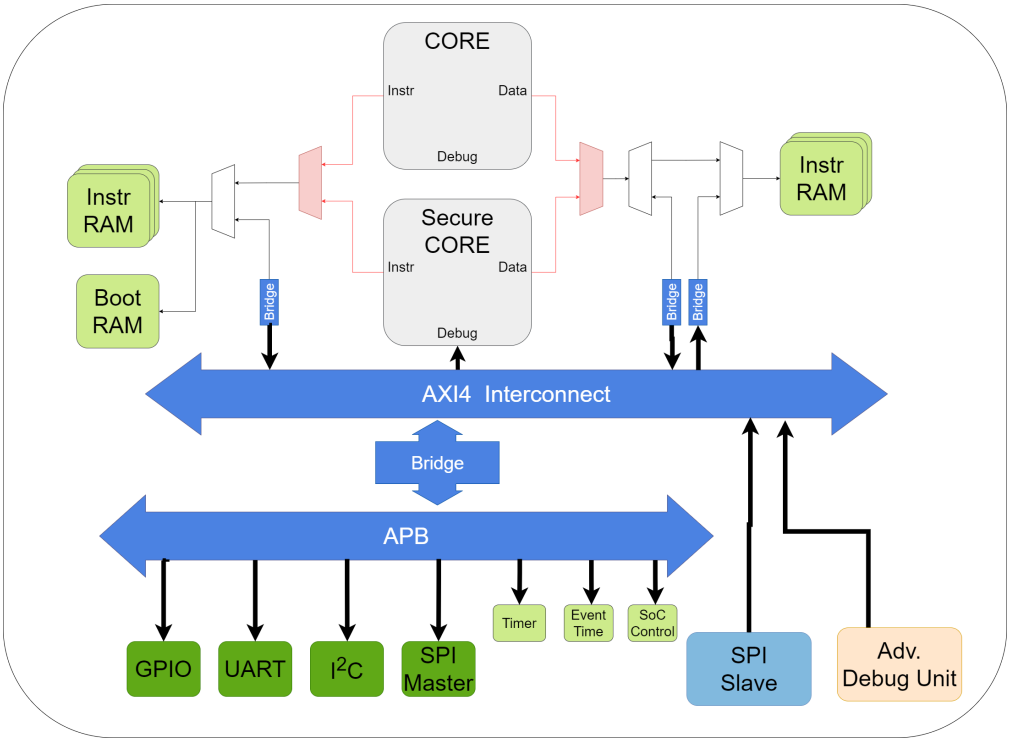

Figure 5.6 : Architecture du cœur du véhicule de test

La seule différence entre les deux cœurs réside dans l'incorporation de ces contre-mesures, ce qui permet une comparaison plus fiable et rigoureuse. Cette configuration dual-cœur

offre l'avantage de faciliter l'évaluation comparative, tant du point de vue de la consommation électrique, chaque cœur disposant de sa propre mesure de courant dédiée, que de la résilience aux attaques, notamment lors des campagnes d'injection de fautes et d'analyses par canaux cachés. Ainsi, les deux cœurs, étant en grande partie identiques, servent de base solide pour une évaluation comparative approfondie.

Intégration des tags homomorphiques

Dans une architecture de CPU pipelinée visant à optimiser les performances temporelles, les données sont initialement chargées dans un banc de registres avant de subir diverses transformations. Chaque donnée x est stockée dans le banc de registres sous la forme $(x, p\alpha(x))$. Ainsi, dans un processeur 32 bits, les données sont manipulées sur 64 bits sous la forme $(x, p\alpha(x))$.

L'architecture sur laquelle notre contre-mesure a été ajoutée est un cœur RISC-V à 4 étages avec exécution dans l'ordre (CV32E40P). Un schéma de cette architecture est présenté en figure 5.7.

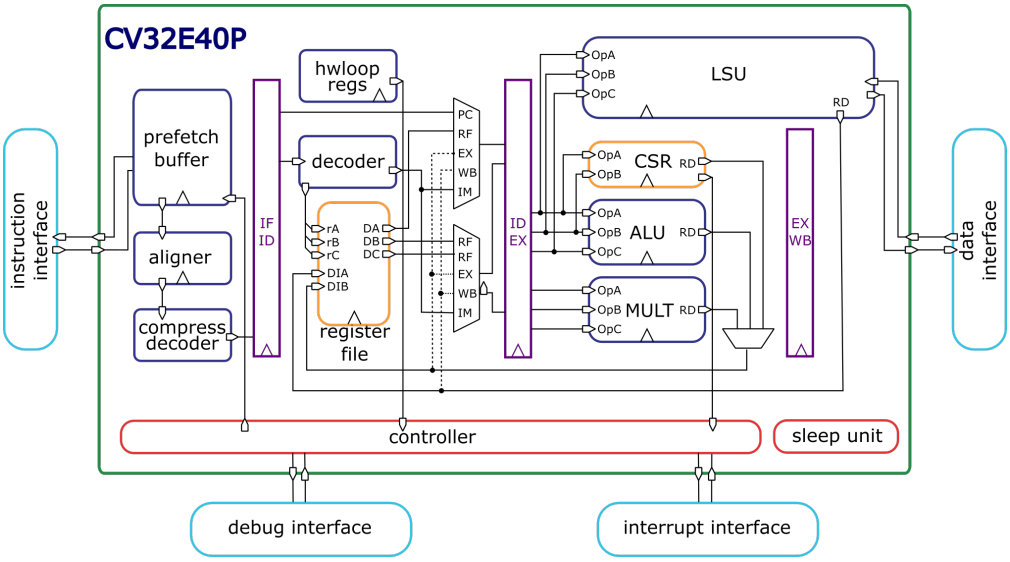

FIGURE 5.7 : Architecture du cœur CV32E40P

Les instructions arrivent à l'étage de FETCH sous la forme (x), bien qu'il soit également possible de les protéger en utilisant des tags homomorphiques, comme discuté précédemment. Les données sont ensuite chargées dans des registres à l'interface mémoire sous la forme $(x, p\alpha(x))$ et peuvent éventuellement être transformées par l'ALU.

L'ALU est capable d'exécuter une variété d'instructions, y compris des opérations logiques pures (AND, NAND, NOR, OR, XOR), des décalages (SHIFT), des additions et soustractions (ADD, SUB), ainsi que des multiplications et divisions (MUL, DIV).

Les calculs sur x et $p\alpha(x)$ sont effectués en parallèle. À la fin de ces calculs, les résultats sont comparés en effectuant une permutation d'un côté ou de l'autre. Cette approche permet de vérifier l'intégrité des données tout au long du pipeline, offrant ainsi une protection supplémentaire contre les attaques par injection de fautes et les attaques par canaux cachés.

5.2.2 Modification du chemin de données

La contre-mesure a été intégrée dans un cœur de processeur RISC-V, complétée par une étape de permutation des données à l'entrée du cœur. Pour accommoder cette modification, tous les registres inter-étages ainsi que le banc de registres ont été adaptés afin de stocker les données dans leur forme permutée.

Quatre points de vérification ont été stratégiquement ajoutés pour assurer l'intégrité des données :

- Trois ports de lecture du banc de registres, qui correspondent à la sortie de l'étage de décodage (DECOD).
- Deux ports d'écriture du banc de registres, qui correspondent à la sortie de l'étage d'exécution (EXEC).
- L'entrée de l'étage d'exécution (EXEC).
- La sortie de l'Unité de Chargement et de Stockage (LSU, Load-Store Unit).

Étant donné que le tag d'intégrité est appliqué dès l'entrée du processeur, il n'est pas nécessaire de procéder à une vérification des données à ce point précis. En règle générale, les données sont soumises à des contrôles d'intégrité à l'entrée et à la sortie de chaque étage où elles subissent des manipulations. Si une divergence est détectée, une exception spécifique est levée. Il convient de souligner que cette exception ne peut être déclenchée que par une faute, qu'elle soit d'origine naturelle (comme une particule chargée) ou malveillante.

En complément des modifications structurelles apportées au cœur du processeur, des ajustements ont été nécessaires pour les contre-mesures déjà en place. Par exemple, le CV32E40P supporte des opérations de décalage et d'addition en modes vectoriels. Cela implique que ces opérations doivent être en mesure de traiter non seulement des données

de 32 bits, mais également des blocs de données de 2x16 bits et de 4x8 bits. Notre solution, qui sépare les données en utilisant une approche dichotomique, permet d'exécuter ces types d'opérations sur les données permutées. En inhibant la propagation à un point spécifique dans nos arbres d'addition et de décalage, nous avons réussi à implémenter ces opérateurs de manière efficace.

5.2.3 Port de test

Afin d'évaluer l'efficacité des instructions factices et de la technique de sécurisation SEC-DEC, un registre de vérification des instructions a été intégré à l'entrée de l'étage de décodage du pipeline du processeur. Ce registre est actualisé à chaque cycle d'horloge, permettant ainsi un suivi exhaustif des instructions qui sont injectées dans le pipeline. Cette approche offre la possibilité de comparer ces instructions avec celles présentes dans le code binaire du programme exécuté, afin de confirmer que l'insertion d'instructions factices ou le masquage d'instructions a été réalisé de manière adéquate.

Par ailleurs, un simulateur de fautes a été incorporé au sein du cœur sécurisé du processeur. Ce simulateur applique un masque de fautes aux instructions ou aux données à l'entrée du pipeline. Cette fonctionnalité permet de simuler des scénarios de fautes et d'évaluer ainsi l'efficacité des contre-mesures implémentées.

Dans le contexte plus spécifique de la gestion des tags d'intégrité, il est possible d'activer individuellement tous les points de vérification. Cette flexibilité permet de valider de manière ciblée le bon fonctionnement des mécanismes de contre-mesure associés aux tags d'intégrité.

Surcoût Matériel

Afin d'évaluer l'impact matériel engendré par ces modifications, des comparaisons ont été menées, en particulier en ce qui concerne les instructions de branchement et de branchement conditionnel. Les multiples redondances sur l'ALU ainsi que les points de vérification et de test ont entraîné une augmentation de 74% de la surface du cœur (cœur considéré isolément, à l'exclusion des bus externes, périphériques, caches et mémoires) atteignant ainsi 25624 GE. Dans notre évaluation, nous prenons en compte que la porte NAND2 occupe une surface de 4 GE. Ce surcoût est manifestement plus conséquent qu'une simple duplication de l'ALU, toutefois, il apporte des garanties plus substantielles. L'intégration des opérations permutées n'a requis aucun cycle de calcul supplémentaire par rapport à

l'ALU canonique. En considérant désormais l'implémentation des trois contre-mesures, nous aboutissons à une surface de 29319 GE, soit une augmentation de 102,5% par rapport au cœur de base. Ces comparaisons permettent de quantifier l'impact de ces contre-mesures sur la complexité du matériel et, par conséquent, sur les coûts de fabrication et de déploiement.

Évaluation

Cette démarche de security assessment a été évaluée de deux façons : premièrement grâce à l'instrumentation du circuit faite à la conception et deuxièmement par des attaques sur un banc laser. Le circuit a en effet été instrumenté pour pouvoir fauter une donnée grâce à un registre qui est xoré avec cette donnée. Cela permet de simuler une faute ou des fautes sur plusieurs bits à plusieurs endroits du circuit permettant de valider la fonctionnalité de la contre-mesure. Concernant les attaques laser, elles ont montré de grandes difficultés à trouver un point de fonctionnement pour l'intensité du laser. La plage d'intensité efficace est en effet très restreinte sur des technologies FDSOI de par la faible épaisseur de la zone active du silicium. Soit l'intensité est trop faible pour générer des fautes, soit elle est trop importante et mène à la destruction du composant. Un point de fonctionnement sûr a toutefois pu être trouvé. En disposant du schéma du circuit du CPU, le backend a été éclairé entraînant de façon systématique soit un crash soit la levée de l'exception qui a été implémentée pour détecter une erreur dans le calcul. Des attaques multifaisceaux ciblant un bit du domaine canonique et le bit associé du domaine permuté n'ont malheureusement pas pu être concluant. Des tentatives d'attaques laser sur le frontend non protégé ont bien montré par contre des comportements non attendus du circuit. Les instructions et leur décodage ne sont effectivement pas sciemment protégés sur cet ASIC. Un point faible subsiste toutefois comme dans de nombreuses contremesures contre les fautes, c'est l'unique point de vérification final. En effet, même si plusieurs points de vérifications ont été implémentés dans le circuit, si la faute est générée avant le dernier point de vérification du datapath et qu'une faute précise trompe le signal d'exception du point de vérification alors une donnée fautée peut passer les différents remparts de protections. Avec notre banc laser, nous n'avons pas réussi à valider une telle attaque demandant deux fautes distinctes mais ce risque doit être pris en compte en implémentant par exemple une duplication des signaux de vérification. En conclusion forcément temporaire dans ce type d'évaluation qui dépend grandement du temps qu'on y passe, nous pouvons affirmer la bonne détection des fautes sur les données dans tout le datapath du circuit.

5.3 Conclusion

En conclusion de ce chapitre, nous avons exposé une solution matérielle innovante pour l'insertion en temps réel d'instructions factices, spécifiquement optimisée pour les processeurs à usage général et engendrant un surcout minimale. Cette approche se distingue par l'insertion d'un délai aléatoire, déterminé en fonction du contexte d'exécution, et par l'utilisation d'instructions factices aléatoires déjà utilisées, sans nécessité de prologue ou de préambule. Cette caractéristique nous permet d'atteindre un niveau d'optimisation que les solutions purement logicielles ne peuvent égaler. De plus, nos instructions factices sont conçues pour être en cohérence avec le contexte d'exécution, offrant ainsi une variabilité élevée.

Des travaux futurs sont nécessaires pour évaluer l'impact de cette variabilité sur les fuites d'informations et, plus cruciallement, pour examiner la faisabilité de la resynchronisation des traces en distinguant les instructions authentiques des instructions factices.

Nous avons également discuté de l'implémentation de diverses contre-mesures dans un processeur RISC-V, ainsi que des défis associés à leur intégration dans un véhicule de test unique. L'utilisation de deux cœurs identiques sur la même puce a permis des campagnes de tests comparatives sur du matériel identique, offrant ainsi une base solide pour l'évaluation de la robustesse du système. Les modifications apportées au chemin de données et les différents mécanismes de test mis en place ont également été abordés.

6

Conclusion

L'état actuel de la recherche a identifié diverses méthodes pour obfusquer le pipeline d'un processeur, mais aucune n'a élaboré de stratégies spécifiques pour neutraliser les attaques par injection de fautes et par canaux auxiliaires. En conséquence, il n'existe pas de travaux académiques qui traitent ces deux défis simultanément.

Dans ce contexte, la présente thèse s'est concentrée sur l'adaptation des contraintes inhérentes à un pipeline de CPU, en mettant en avant des solutions novatrices. Une attention particulière a été portée à l'impact matériel des solutions envisagées pour assurer une intégration économiquement viable. Outre la question du coût, l'adaptabilité et la généralisation à diverses architectures ont été des éléments centraux de cette démarche de recherche, visant à permettre une intégration sur un large spectre de cibles, depuis le microcontrôleur à exécution séquentielle jusqu'au processeur d'application à l'exécution spéculative.

La stratégie adoptée a été de se focaliser sur les composants fondamentaux communs à tous les processeurs, à savoir le chemin de contrôle et le chemin de données. Deux solutions ont été avancées pour le chemin de données.

La première implique l'ajout de tags d'intégrité, générées par le biais d'une permutation aléatoire. Cette permutation aléatoire accroît la robustesse du système face aux attaques par injection de fautes. En raison de la nature aléatoire de la permutation, un adversaire, même doté d'une connaissance complète du système, ne peut garantir la réussite d'une injection de faute.

Enfin, l'utilisation de registres dynamiques est proposée. Bien que déjà présents dans la littérature existante, ces registres confèrent une sécurité accrue en randomisant l'empla-

cement des données. Ces registres constituent en effet le maillon le plus vulnérable du chemin de données et interagissent également de manière cruciale avec la contre-mesure des cycles factices.

Pour le chemin de contrôle, des mesures de sécurité plus légères ont été employées pour contrer les canaux auxiliaires, étant donné sa moindre sensibilité à ce type d'attaques. Les mesures de sécurité se sont principalement axées sur la création de dépendances entre les instructions. La solution avancée consiste à générer un masque à partir des signaux de décodage de l'instruction précédente pour masquer l'instruction en cours. De plus, les signaux de contrôle sont sécurisés grâce à une parité croisée, offrant une protection substantielle contre les injections de fautes multiples à un coût marginal.

D'un point de vue global, un système de désynchronisation temporelle aléatoire a été introduit pour compliquer les attaques par injection de fautes et par canaux auxiliaires. Cette méthode implique l'insertion d'instructions aléatoires indiscernables des instructions légitimes.

Les diverses contre-mesures ont été implantées dans un véhicule de test qui a subi un processus de fonderie et sera évalué par le CESTI du CEA-LETI.

Cette thèse a conduit à plusieurs conclusions notables. La première est que l'introduction de l'aléatoire en réponse aux attaques par erreur est la méthode préventive la plus efficace. En effet, un adversaire, même doté d'une connaissance exhaustive du système cible, serait incapable de mener une attaque réussie. De plus, cette stratégie contribue à introduire de l'aléa dans les traces pour les canaux auxiliaires, rendant ainsi l'analyse statistique plus ardue.

Par ailleurs, l'approche de masquage et de détection via le décodage des instructions fournit des solutions extrêmement légères pour la détection des erreurs au niveau des instructions.

En termes de résilience, la gestion des attaques par injection de fautes est principalement du ressort du logiciel, via des indicateurs spécifiques. Aborder ces cas rares à partir d'une perspective matérielle s'avère souvent trop complexe. La correction d'erreur demeure la solution la plus simple, bien qu'elle crée de nouvelles opportunités pour les attaques. Néanmoins, en ajoutant l'authenticité aux propriétés des contre-mesures pour les attaques par injection de fautes, il est possible d'atteindre un équilibre entre sécurité et sûreté sans compromettre l'un ou l'autre.

Ainsi, cette thèse apporte un ensemble de contributions significatives pour la protection des éléments clés d'un pipeline de processeurs contre les attaques par canaux auxiliaires et par injection de fautes.

6.1 Perspectives

Il demeure plusieurs axes de recherche à explorer, notamment en matière de résilience. Bien que des indicateurs spécifiques pour les erreurs matérielles aient été instaurés, il est impératif que le logiciel puisse gérer ces cas avec des procédures adaptées. Le traitement d'une faute au niveau du registre, qui s'avère être un défi majeur, constitue l'un des problèmes les plus épineux identifiés dans cette thèse.

La question de la génération de nombres aléatoires, qui n'a pas été abordée dans ce travail, reste un point en suspens. L'efficacité des mécanismes basés sur l'aléatoire dépend en grande partie de la qualité de la source d'aléa, ce qui mérite une investigation plus approfondie.

Des recherches complémentaires sur l'optimisation des tags d'intégrité, notamment en ce qui concerne les opérations de décalage et de multiplication, sont essentielles.

En ce qui concerne SECDEC, il serait bénéfique de le perfectionner pour en faire un contrôle de flux d'intégrité plus complet et plus efficient que les solutions actuellement disponibles. Cela pourrait inclure des méthodes pour réduire la latence ou le surcoût matériel associé à cette technique.

Enfin, ces contre-mesures ayant été intégrées dans un processeur 32 bits, il serait logique de les tester sur des architectures plus sophistiquées. Les optimisations spécifiques à ces architectures pourraient interagir de manière intéressante avec les contre-mesures proposées, offrant ainsi de nouvelles opportunités d'amélioration. Par exemple, une étude sur les interactions entre les cycles factices et les mécanismes d'exécution désordonnée pourrait révéler des synergies ou des conflits potentiels qui mériteraient une attention particulière.

Ces perspectives ouvrent la voie à des recherches futures qui pourraient non seulement valider l'efficacité des solutions proposées mais aussi les améliorer ou les étendre à de nouveaux contextes.

BIBLIOGRAPHIE

[Ago+10] Michel AGOYAN et al., "How to flip a bit?", in : *2010 IEEE 16th International On-Line Testing Symposium*, ISSN : 1942-9401, juill. 2010, p. 235-239, DOI : 10.1109/IOLTS.2010.5560194.

[ALF02] L. ANTONI, R. LEVEUGLE et M. FEHER, "Using run-time reconfiguration for fault injection in hardware prototypes", in : *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings.* ISSN : 1550-5774, nov. 2002, p. 245-253, DOI : 10.1109/DFTVS.2002.1173521.

[Arl+90] J. ARLAT et al., "Fault injection for dependability validation : a methodology and some applications", in : *IEEE Transactions on Software Engineering* **16**.2 (fév. 1990), Conference Name : IEEE Transactions on Software Engineering, p. 166-182, ISSN : 1939-3520, DOI : 10.1109/32.44380.

[ARP12] Jude Angelo AMBROSE, Roshan G. RAGEL et Sri PARAMESWARAN, "Randomized Instruction Injection to Counter Power Analysis Attacks", in : *ACM Trans. Embed. Comput. Syst.* **11**.3 (2012), 69 :1-69 :28, DOI : 10.1145/2345770.2345782, URL : https://doi.org/10.1145/2345770.2345782.

[Bar+15] Gilles BARTHE et al., "Verified Proofs of Higher-Order Masking", in : *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, sous la dir. d'Elisabeth OSWALD et Marc FISCHLIN, t. 9056, Lecture Notes in Computer Science, Springer, 2015, p. 457-485, DOI : 10.1007/978-3-662-46800-5_18, URL : https://doi.org/10.1007/978-3-662-46800-5_18.

[Bar+16a] Gilles BARTHE et al., "Strong Non-Interference and Type-Directed Higher-Order Masking", in : *CCS 2016 - 23rd ACM Conference on Computer and Communications Security*, Vienne, Austria : ACM, oct. 2016, p. 116 -129, DOI : 10.1145/2976749.2978427, URL : https://hal.inria.fr/hal-01410216.

[Bar+16b] Gilles BARTHE et al., "Strong Non-Interference and Type-Directed Higher-Order Masking", in : *Proceedings of the 2016 ACM SIGSAC Conference on Com-*

puter and Communications Security, Vienna, Austria, October 24-28, 2016, sous la dir. d'Edgar R. Weippl et al., ACM, 2016, p. 116-129, DOI : 10.1145/2976749.2978427, URL : https://doi.org/10.1145/2976749.2978427.

[Bar+21] Marcello Barbirotta et al., "A Fault Tolerant soft-core obtained from an Interleaved-Multi- Threading RISC- V microprocessor design", in : *36th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2021, Athens, Greece, October 6-8, 2021*, sous la dir. de Luigi Dilillo, Luca Cassano et Athanasios Papadimitriou, IEEE, 2021, p. 1-4, DOI : 10.1109/DFT52944.2021.9568368, URL : https://doi.org/10.1109/DFT52944.2021.9568368.

[Ber05] Daniel J. Bernstein, "Cache-timing attacks on AES", en, in : (2005), URL : https://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.140.2501.

[BFP19] Claudio Bozzato, Riccardo Focardi et Francesco Palmarini, "Shaping the Glitch : Optimizing Voltage Fault Injection Attacks", en, in : *IACR Transactions on Cryptographic Hardware and Embedded Systems* (fév. 2019), p. 199-224, ISSN : 2569-2925, DOI : 10.13154/tches.v2019.i2.199-224, URL : https://tches.iacr.org/index.php/TCHES/article/view/7390.

[BGV11] Josep Balasch, Benedikt Gierlichs et Ingrid Verbauwhede, "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs", in : *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, sept. 2011, p. 105-114, DOI : 10.1109/FDTC.2011.9.

[BJ15] Jakub Breier et Dirmanto Jap, "Testing Feasibility of Back-Side Laser Fault Injection on a Microcontroller", in : *Proceedings of the WESS'15 : Workshop on Embedded Systems Security*, WESS'15, New York, NY, USA : Association for Computing Machinery, oct. 2015, ISBN : 978-1-4503-3667-3, DOI : 10.1145/2818362.2818367, URL : https://doi.org/10.1145/2818362.2818367.

[Bla+19] Luigi Blasi et al., "A RISC-V Fault-Tolerant Microcontroller Core Architecture Based on a Hardware Thread Full/Partial Protection and a Thread-Controlled Watch-Dog Timer", in : *Applications in Electronics Pervading Industry, Environment and Society - APPLEPIES 2019, Pisa, Italy, 11-13 September 2019*, sous la dir. de Sergio Saponara et Alessandro De Gloria, t. 627, Lecture Notes in Electrical Engineering, Springer, 2019, p. 505-511, DOI : 10.1007/978-3-030-37277-4_59, URL : https://doi.org/10.1007/978-3-030-37277-4_59.

[BS97] Eli Biham et Adi Shamir, "Differential fault analysis of secret key cryptosystems", en, in : *Advances in Cryptology — CRYPTO '97*, sous la dir. de Burton S. Kaliski, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 1997, p. 513-525, isbn : 978-3-540-69528-8, doi : 10.1007/BFb0052259.

[Buc+06] Marco Bucci et al., "Three-Phase Dual-Rail Pre-charge Logic", en, in : *Cryptographic Hardware and Embedded Systems - CHES 2006*, sous la dir. de Louis Goubin et Mitsuru Matsui, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 2006, p. 232-241, isbn : 978-3-540-46561-4, doi : 10.1007/11894063_19.

[Cag18] Eleonora Cagli, "Feature Extraction for Side-Channel Attacks. (Extraction de caractéristiques pour les attaques par canaux auxiliaires)", thèse de doct., Sorbonne University, France, 2018, url : https://tel.archives-ouvertes.fr/tel-02494260.

[CCD00] Christophe Clavier, Jean-Sébastien Coron et Nora Dabbous, "Differential Power Analysis in the Presence of Hardware Countermeasures", in : *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, sous la dir. de Çetin Kaya Koç et Christof Paar, t. 1965, Lecture Notes in Computer Science, Springer, 2000, p. 252-263, doi : 10.1007/3-540-44499-8_20, url : https://doi.org/10.1007/3-540-44499-8_20.

[CK09] Jean-Sébastien Coron et Ilya Kizhvatov, "An Efficient Method for Random Delay Generation in Embedded Software", in : *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, sous la dir. de Christophe Clavier et Kris Gaj, t. 5747, Lecture Notes in Computer Science, Springer, 2009, p. 156-170, doi : 10.1007/978-3-642-04138-9_12, url : https://doi.org/10.1007/978-3-642-04138-9_12.

[CLH19] Valence Cristiani, Maxime Lecomte et Thomas Hiscock, "A Bit-Level Approach to Side Channel Based Disassembling", in : *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, sous la dir. de Sonia Belaïd et Tim Güneysu, t. 11833, Lecture Notes in Computer Science, Springer, 2019, p. 143-158, doi : 10.1007/978-3-030-42068-0_9, url : https://doi.org/10.1007/978-3-030-42068-0_9.

[Col+21] Brice Colombier et al., "Multi-Spot Laser Fault Injection Setup : New Possibilities for Fault Injection Attacks", in : *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, sous la dir. de Vincent Grosso et Thomas Pöppelmann, t. 13173, Lecture Notes in Computer Science, Springer, 2021, p. 151-166, doi : 10.1007/978-3-030-97348-3_9, url : https://doi.org/10.1007/978-3-030-97348-3_9.

[CS20] Gaëtan Cassiers et François-Xavier Standaert, "Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference", in : *IEEE Trans. Inf. Forensics Secur.* **15** (2020), p. 2542-2555, doi : 10.1109/TIFS.2020.2971153, url : https://doi.org/10.1109/TIFS.2020.2971153.

[CV17] Ruan de Clercq et Ingrid Verbauwhede, "A survey of Hardware-based Control Flow Integrity (CFI)", in : *CoRR* **abs/1706.07257** (2017), arXiv : 1706.07257, url : http://arxiv.org/abs/1706.07257.

[CZ06] Zhimin Chen et Yujie Zhou, "Dual-Rail Random Switching Logic : A Countermeasure to Reduce Side Channel Leakage", en, in : *Cryptographic Hardware and Embedded Systems - CHES 2006*, sous la dir. de Louis Goubin et Mitsuru Matsui, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 2006, p. 242-254, isbn : 978-3-540-46561-4, doi : 10.1007/11894063_20.

[Dad65] Luigi Dadda, "Some schemes for parallel multipliers", in : *Alta frequenza* **34** (1965), p. 349-356.

[Dar+01] F. Darracq et al., "Single-event sensitivity of a single SRAM cell", in : *RADECS 2001. 2001 6th European Conference on Radiation and Its Effects on Components and Systems* (*Cat. No.01TH8605*), sept. 2001, p. 387-391, doi : 10.1109/RADECS.2001.1159311.

[Deh+12] Amine Dehbaoui et al., "Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES", in : *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, sept. 2012, p. 7-15, doi : 10.1109/FDTC.2012.15.

[DLM19] Mathieu Dumont, Mathieu Lisart et Philippe Maurine, "Electromagnetic Fault Injection : How Faults Occur", in : *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography* (*FDTC*), août 2019, p. 9-16, doi : 10.1109/FDTC.2019.00010.

[DOCB06] A. Djellid-Ouar, G. Cathebras et F. Bancel, "Supply voltage glitches effects on CMOS circuits", in : *International Conference on Design and Test of Integrated*

Systems in Nanoscale Technology, 2006. DTIS 2006. Sept. 2006, p. 257-261, DOI : 10.1109/DTIS.2006.1708651.

[Dur+12] François DURVAUX et al., "Efficient Removal of Random Delays from Embedded Software Implementations Using Hidden Markov Models", in : *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, sous la dir. de Stefan MANGARD, t. 7771, Lecture Notes in Computer Science, Springer, 2012, p. 123-140, DOI : 10.1007/978-3-642-37288-9_9, URL : https://doi.org/10.1007/978-3-642-37288-9_9.

[Dut+18] Jean-Max DUTERTRE et al., "Laser Fault Injection at the CMOS 28 nm Technology Node : an Analysis of the Fault Model", in : *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, sept. 2018, p. 1-6, DOI : 10.1109/FDTC.2018.00009.

[Dut+19] Jean-Max DUTERTRE et al., "Experimental Analysis of the Laser-Induced Instruction Skip Fault Model", in : *Secure IT Systems - 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18-20, 2019, Proceedings*, sous la dir. d'Aslan ASKAROV, René Rydhof HANSEN et Willard RAFNSSON, t. 11875, Lecture Notes in Computer Science, Springer, 2019, p. 221-237, DOI : 10.1007/978-3-030-35055-0_14, URL : https://doi.org/10.1007/978-3-030-35055-0_14.

[Ent+09] Luis ENTRENA et al., "SET Emulation Considering Electrical Masking Effects", in : *IEEE Transactions on Nuclear Science* **56**.4 (août 2009), Conference Name : IEEE Transactions on Nuclear Science, p. 2021-2025, ISSN : 1558-1578, DOI : 10.1109/TNS.2009.2013346.

[Esl+20] Mohammad ESLAMI et al., "A survey on fault injection methods of digital integrated circuits", en, in : *Integration* **71** (mars 2020), p. 154-163, ISSN : 0167-9260, DOI : 10.1016/j.vlsi.2019.11.006, URL : https://www.sciencedirect.com/science/article/pii/S016792601930402X.

[Fib+19] Christian FIBICH et al., "FIJI : Fault InJection Instrumenter", in : *EURASIP Journal on Embedded Systems* **2019** (déc. 2019), DOI : 10.1186/s13639-019-0088-7.

[GM13] Michael GRUHN et Tilo MÜLLER, "On the Practicability of Cold Boot Attacks", in : *2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2-6, 2013*, IEEE Computer Society, 2013, p. 390-397, DOI : 10.1109/ARES.2013.52, URL : https://doi.org/10.1109/ARES.2013.52.

[GM18] Hannes Gross et Stefan Mangard, "A unified masking approach", in : *J. Cryptogr. Eng.* **8**.*2* (2018), p. 109-124, doi : 10.1007/s13389-018-0184-y, url : https://doi.org/10.1007/s13389-018-0184-y.

[GMK16] Hannes Gross, Stefan Mangard et Thomas Korak, "Domain-Oriented Masking : Compact Masked Hardware Implementations with Arbitrary Protection Order", in : *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, sous la dir. de Begül Bilgin, Svetla Nikova et Vincent Rijmen, ACM, 2016, p. 3, doi : 10.1145/2996366.2996426, url : https://doi.org/10.1145/2996366.2996426.

[GMK17] Hannes Gross, Stefan Mangard et Thomas Korak, "An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order", in : *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, sous la dir. d'Helena Handschuh, t. 10159, Lecture Notes in Computer Science, Springer, 2017, p. 95-112, doi : 10.1007/978-3-319-52153-4_6, url : https://doi.org/10.1007/978-3-319-52153-4_6.

[God+09] C. Godlewski et al., "Electrical modeling of the effect of beam profile for pulsed laser fault injection", en, in : *Microelectronics Reliability*, 20th European Symposium on the Reliability of Electron Devices, Failure Physics and Analysis **49**.*9* (sept. 2009), p. 1143-1147, issn : 0026-2714, doi : 10.1016/j.microrel.2009.07.037, url : https://www.sciencedirect.com/science/article/pii/S0026271409002674.

[Goo+11] G. Goodwill et al., *A testing methodology for side channel resistance*, en, 2011.

[Gro18] Hannes Gross, "Domain-Oriented Masking : Generically Masked Hardware Implementations", English, thèse de doct., Graz University of Technology (90000), 2018.

[GST17] Daniel Genkin, Adi Shamir et Eran Tromer, "Acoustic Cryptanalysis", en, in : *J Cryptol* **30**.*2* (avr. 2017), p. 392-443, issn : 1432-1378, doi : 10.1007/s00145-015-9224-2, url : https://doi.org/10.1007/s00145-015-9224-2.

[Gui+11] Sylvain Guilley et al., "Formal Framework for the Evaluation of Waveform Resynchronization Algorithms", in : *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2 International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings*, sous la dir. de Claudio A. Ardagna et Jianying Zhou, t. 6633, Lecture Notes in Computer Science, Springer, 2011, p. 100-115, doi : 10.1007/

978-3-642-21040-2_7, URL : https://doi.org/10.1007/978-3-642-21040-2_7.

[Hal+09] J. Alex HALDERMAN et al., "Lest we remember : cold-boot attacks on encryption keys", in : *Commun. ACM* **52**.*5* (mai 2009), p. 91-98, ISSN : 0001-0782, DOI : 10.1145/1506409.1506429, URL : https://doi.org/10.1145/1506409.1506429.

[He+16] Zhangqing HE et al., "ERIST : An Efficient Randomized Instruction Insertion Technique to Counter Side-Channel Attacks", in : *IAENG International Journal of Computer Science* **43** (fév. 2016), p. 65-71.

[HS13a] Michael HUTTER et Jörn-Marc SCHMIDT, "The Temperature Side-Channel and Heating Fault Attacks", in : t. 8419, nov. 2013, DOI : 10.1007/978-3-319-08302-5_15.

[HS13b] Michael HUTTER et Jörn-Marc SCHMIDT, "The Temperature Side-Channel and Heating Fault Attacks", in : t. 8419, nov. 2013, DOI : 10.1007/978-3-319-08302-5_15.

[ISW03a] Yuval ISHAI, Amit SAHAI et David WAGNER, "Private Circuits : Securing Hardware against Probing Attacks", en, in : *Advances in Cryptology - CRYPTO 2003*, sous la dir. de Dan BONEH, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 2003, p. 463-481, ISBN : 978-3-540-45146-4, DOI : 10.1007/978-3-540-45146-4_27.

[ISW03b] Yuval ISHAI, Amit SAHAI et David A. WAGNER, "Private Circuits : Securing Hardware against Probing Attacks", in : *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, sous la dir. de Dan BONEH, t. 2729, Lecture Notes in Computer Science, Springer, 2003, p. 463-481, DOI : 10.1007/978-3-540-45146-4_27, URL : https://doi.org/10.1007/978-3-540-45146-4_27.

[KDN14] Maha KOOLI et Giorgio DI NATALE, "A survey on simulation-based fault injection tools for complex systems", in : *2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, mai 2014, p. 1-6, DOI : 10.1109/DTIS.2014.6850649.

[KJJ99] Paul KOCHER, Joshua JAFFE et Benjamin JUN, "Differential Power Analysis", en, in : *Advances in Cryptology — CRYPTO' 99*, sous la dir. de Michael WIENER, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 1999, p. 388-397, ISBN : 978-3-540-48405-9, DOI : 10.1007/3-540-48405-1_25.

[KJP14] Raghavan Kumar, Philipp Jovanovic et Ilia Polian, "Precise Fault-Injections using Voltage and Temperature Manipulation for Differential Cryptanalysis", in : juill. 2014, p. 43-48, isbn : 978-1-4799-5324-0, doi : 10.1109/IOLTS.2014.6873670.

[Kni+21] David Knichel et al., *Automated Generation of Masked Hardware*, Cryptology ePrint Archive, Paper 2021/569, https://eprint.iacr.org/2021/569, 2021, url : https://eprint.iacr.org/2021/569.

[KO63] Anatolij A. Karatsuba et Yu. Ofman, "Multiplication of Multidigit Numbers on Automata", in : *Soviet physics. Doklady* **7** (1963), p. 595-596, url : https://api.semanticscholar.org/CorpusID:117858583.

[Koc96] Paul C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", en, in : *Advances in Cryptology — CRYPTO '96*, sous la dir. de Neal Koblitz, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 1996, p. 104-113, isbn : 978-3-540-68697-2, doi : 10.1007/3-540-68697-5_9.

[Kog+97] R. Koga et al., "Single event functional interrupt (SEFI) sensitivity in microcircuits", in : *RADECS 97. Fourth European Conference on Radiation and its Effects on Components and Systems (Cat. No.97TH8294)*, sept. 1997, p. 311-318, doi : 10.1109/RADECS.1997.698915.

[Kor16] Roman Korkikian, "Side-channel and fault analysis in the presence of countermeasures : tools, theory, and practice", en, thèse de doct., Université Paris sciences et lettres, oct. 2016, url : https://tel.archives-ouvertes.fr/tel-01762404.

[LMW14] Andrew J. Leiserson, Mark E. Marson et Megan A. Wachs, "Gate-Level Masking under a Path-Based Leakage Metric", in : *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, sous la dir. de Lejla Batina et Matthew Robshaw, t. 8731, Lecture Notes in Computer Science, Springer, 2014, p. 580-597, doi : 10.1007/978-3-662-44709-3_32, url : https://doi.org/10.1007/978-3-662-44709-3_32.

[Lon+15] Jake Longo et al., "SoC It to EM : ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip", in : *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, sous la dir. de Tim Güneysu et Helena Handschuh, t. 9293, Lecture Notes in Computer Science, Springer, 2015, p. 620-640, doi : 10.1007/

978-3-662-48324-4_31, URL : https://doi.org/10.1007/978-3-662-48324-4_31.

[LSB22a] Gaëtan Leplus, Olivier Savry et Lilian Bossuet, "Insertion of random delay with context-aware dummy instructions generator in a RISC-V processor", in : *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2022, McLean, VA, USA, June 27-30, 2022*, IEEE, 2022, p. 81-84, DOI : 10.1109/HOST54066.2022.9840060, URL : https://doi.org/10.1109/HOST54066.2022.9840060.

[LSB22b] Gaëtan Leplus, Olivier Savry et Lilian Bossuet, "SecDec : Secure Decode Stage thanks to masking of instructions with the generated signals", in : *25th Euromicro Conference on Digital System Design, DSD 2022, Maspalomas, Spain, August 31 - Sept. 2, 2022*, IEEE, 2022, p. 556-563, DOI : 10.1109/DSD57027.2022.00080, URL : https://doi.org/10.1109/DSD57027.2022.00080.

[Mad+94] Henrique Madeira et al., "RIFLE : A general purpose pin-level fault injector", in : jan. 1994, p. 199-216, ISBN : 978-3-540-58426-1, DOI : 10.1007/3-540-58426-9_132.

[Man04] Stefan Mangard, "Hardware Countermeasures against DPA ? A Statistical Analysis of Their Effectiveness", in : *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, sous la dir. de Tatsuaki Okamoto, t. 2964, Lecture Notes in Computer Science, Springer, 2004, p. 222-235, DOI : 10.1007/978-3-540-24660-2_18, URL : https://doi.org/10.1007/978-3-540-24660-2_18.

[MBR19] Lauren De Meyer, Begül Bilgin et Oscar Reparaz, "Consolidating Security Notions in Hardware Masking", en, in : *IACR Transactions on Cryptographic Hardware and Embedded Systems* (mai 2019), p. 119-147, ISSN : 2569-2925, DOI : 10.13154/tches.v2019.i3.119-147, URL : https://tches.iacr.org/index.php/TCHES/article/view/8291.

[Men+20] Alexandre Menu et al., "Experimental Analysis of the Electromagnetic Instruction Skip Fault Model", in : *15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2020, Marrakech, Morocco, April 1-3, 2020*, IEEE, 2020, p. 1-7, DOI : 10.1109/DTIS48698.2020.9081261, URL : https://doi.org/10.1109/DTIS48698.2020.9081261.

[Mey+19] Lauren De Meyer et al., "M&M : Masks and Macs against Physical Attacks", in : *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**.*1* (2019), p. 25-50, DOI :

10.13154/tches.v2019.i1.25-50, URL : https://doi.org/10.13154/tches.v2019.i1.25-50.

[MMS01] David May, Henk Muller et Nigel Smart, "Non-deterministic Processors", in : t. 2119, juill. 2001, p. 115-129, ISBN : 978-3-540-42300-3, DOI : 10.1007/3-540-47719-5_11.

[Mor+14] Nicolas Moro et al., "Formal verification of a software countermeasure against instruction skip attacks", in : *J. Cryptogr. Eng.* **4**.*3* (2014), p. 145-156, DOI : 10.1007/s13389-014-0077-7, URL : https://doi.org/10.1007/s13389-014-0077-7.

[MPG05] Stefan Mangard, Thomas Popp et Berndt M. Gammel, "Side-Channel Leakage of Masked CMOS Gates", in : *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, sous la dir. d'Alfred Menezes, t. 3376, Lecture Notes in Computer Science, Springer, 2005, p. 351-365, DOI : 10.1007/978-3-540-30574-3_24, URL : https://doi.org/10.1007/978-3-540-30574-3_24.

[MRB18] Lauren De Meyer, Oscar Reparaz et Begül Bilgin, "Multiplicative Masking for AES in Hardware", en, in : *IACR Transactions on Cryptographic Hardware and Embedded Systems* (août 2018), p. 431-468, ISSN : 2569-2925, DOI : 10.13154/tches.v2018.i3.431-468, URL : https://tches.iacr.org/index.php/TCHES/article/view/7282.

[Nag+22] Rishub Nagpal et al., "Riding the Waves Towards Generic Single-Cycle Masking in Hardware", in : *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**.*4* (2022), p. 693-717, DOI : 10.46586/tches.v2022.i4.693-717, URL : https://doi.org/10.46586/tches.v2022.i4.693-717.

[Nav+11] L. A. B. Naviner et al., "FIFA : A fault-injection–fault-analysis-based tool for reliability assessment at RTL level", en, in : *Microelectronics Reliability*, Proceedings of the 22th European Symposium on the RELIABILITY OF ELECTRON DEVICES, FAILURE PHYSICS AND ANALYSIS **51**.*9* (sept. 2011), p. 1459-1463, ISSN : 0026-2714, DOI : 10.1016/j.microrel.2011.06.017, URL : https://www.sciencedirect.com/science/article/pii/S0026271411002162.

[NRR06] Svetla Nikova, Christian Rechberger et Vincent Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches", in : *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, sous la dir. de Peng Ning, Sihan Qing et Ninghui Li, t. 4307, Lecture Notes in Computer Science, Springer, 2006,

p. 529-545, DOI : 10.1007/11935308_38, URL : https://doi.org/10.1007/11935308_38.

[OGSM17] Sébastien Ordas, Ludovic Guillaume-Sage et Philippe Maurine, "Electromagnetic fault injection : the curse of flip-flops", in : *Journal of Cryptographic Engineering* **7**.*3* (2017), Publisher : Springer, p. 183-197, DOI : 10.1007/s13389-016-0128-3, URL : https://hal-lirmm.ccsd.cnrs.fr/lirmm-01430913.

[Ott05] Martin Otto, *Fault attacks and countermeasures*, 2005.

[Pag+11] Samuel Pagliarini et al., "Analyzing the Impact of Single-Event-Induced Charge Sharing in Complex Circuits", in : *IEEE Transactions on Nuclear Science* **58**.*6* (déc. 2011), Conference Name : IEEE Transactions on Nuclear Science, p. 2768-2775, ISSN : 1558-1578, DOI : 10.1109/TNS.2011.2168239.

[PM05] Thomas Popp et Stefan Mangard, "Masked Dual-Rail Pre-charge Logic : DPA-Resistance Without Routing Constraints", en, in : *Cryptographic Hardware and Embedded Systems – CHES 2005*, sous la dir. de Josyula R. Rao et Berk Sunar, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 2005, p. 172-186, ISBN : 978-3-540-31940-5, DOI : 10.1007/11545262_13.

[QS02] Jean-Jacques Quisquater et D. Samyde, "Eddy current for magnetic analysis with active sensor, Proceedings of ESmart", in : *Proceedings of ESmart* (2002), pp.185-194.

[SA03] Sergei P. Skorobogatov et Ross J. Anderson, "Optical Fault Induction Attacks", en, in : *Cryptographic Hardware and Embedded Systems - CHES 2002*, sous la dir. de Burton S. Kaliski, çetin K. Koç et Christof Paar, Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 2003, p. 2-12, ISBN : 978-3-540-36400-9, DOI : 10.1007/3-540-36400-5_2.

[SEH20] Olivier Savry, Mustapha El-Majihi et Thomas Hiscock, "Confidaent : Control FLow protection with Instruction and Data Authenticated Encryption", in : *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020*, IEEE, 2020, p. 246-253, DOI : 10.1109/DSD51259.2020.00048, URL : https://doi.org/10.1109/DSD51259.2020.00048.

[Sel+16] Bodo Selmke et al., "Precise Laser Fault Injections into 90 nm and 45 nm SRAM-cells", en, in : *Smart Card Research and Advanced Applications*, sous la dir. de Naofumi Homma et Marcel Medwed, Lecture Notes in Computer Science, Cham : Springer International Publishing, 2016, p. 193-205, ISBN : 978-3-319-31271-2, DOI : 10.1007/978-3-319-31271-2_12.

[SH07] Jörn marc Schmidt et Michael Hutter, “Optical and EM Fault-Attacks on CRT-based RSA : Concrete Results”, in : 2007.

[Shi+11] Kyoji Shibutani et al., “Piccolo : An Ultra-Lightweight Blockcipher”, in : *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, sous la dir. de Bart Preneel et Tsuyoshi Takagi, t. 6917, Lecture Notes in Computer Science, Springer, 2011, p. 342-357, doi : 10.1007/978-3-642-23951-9_23, url : https://doi.org/10.1007/978-3-642-23951-9_23.

[Spr01] P. L. Springer, “Analysis of application behavior during fault injection”, in : NTRS Author Affiliations : NTRS Document ID : 20060033756 NTRS Research Center : Jet Propulsion Laboratory (JPL), juill. 2001, url : https://ntrs.nasa.gov/citations/20060033756.

[SSI04] Daisuke Suzuki, Minoru Saeki et Tetsuya Ichikawa, “Random Switching Logic : A Countermeasure against DPA based on Transition Probability.”, in : *IACR Cryptology ePrint Archive* **2004** (jan. 2004), p. 346.

[STB97] V. Sieh, O. Tschache et F. Balbach, “VERIFY : evaluation of reliability using VHDL-models with embedded fault descriptions”, in : *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, ISSN : 0731-3071, juin 1997, p. 32-36, doi : 10.1109/FTCS.1997.614074.

[Sug19] Takeshi Sugawara, “3-Share Threshold Implementation of AES S-box without Fresh Randomness”, in : *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**.*1* (2019), p. 123-145, doi : 10.13154/tches.v2019.i1.123-145, url : https://doi.org/10.13154/tches.v2019.i1.123-145.

[Tal+22] Ezinam Bertrand Talaki et al., “A Memory Hierarchy Protected against Side-Channel Attacks”, in : *Cryptogr.* **6**.*2* (2022), p. 19, doi : 10.3390/cryptography6020019, url : https://doi.org/10.3390/cryptography6020019.

[Tan+12] Ming Tang et al., “Power Analysis Based Reverse Engineering on the Secret Round Function of Block Ciphers”, en, in : *Data and Knowledge Engineering*, sous la dir. d’Yang Xiang et al., Lecture Notes in Computer Science, Berlin, Heidelberg : Springer, 2012, p. 175-188, isbn : 978-3-642-34679-8, doi : 10.1007/978-3-642-34679-8_17.

[TV04] Kris Tiri et Ingrid Verbauwhede, “A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation”, in : *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20*

February 2004, Paris, France, IEEE Computer Society, 2004, p. 246-251, DOI : 10.1109/DATE.2004.1268856, URL : https://doi.org/10.1109/DATE.2004.1268856.

[Val+07] Mario Garcia VALDERAS et al., "SET Emulation Under a Quantized Delay Model", in : *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, ISSN : 2377-7966, sept. 2007, p. 68-78, DOI : 10.1109/DFT.2007.49.

[Wer+18] Mario WERNER et al., "Sponge-Based Control-Flow Protection for IoT Devices", in : *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, IEEE, 2018, p. 214-226, DOI : 10.1109/EuroSP.2018.00023, URL : https://doi.org/10.1109/EuroSP.2018.00023.

[WWB11] Jasper G. J. van WOUDENBERG, Marc F. WITTEMAN et Bram BAKKER, "Improving Differential Power Analysis by Elastic Alignment", in : *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, sous la dir. d'Aggelos KIAYIAS, t. 6558, Lecture Notes in Computer Science, Springer, 2011, p. 104-119, DOI : 10.1007/978-3-642-19074-2_8, URL : https://doi.org/10.1007/978-3-642-19074-2_8.

[ZES89] J.A. ZOUTENDYK, L.D. EDMONDS et L.S. SMITH, "Characterization of multiple-bit errors from single-ion tracks in integrated circuits", in : *IEEE Transactions on Nuclear Science* **36**.*6* (déc. 1989), Conference Name : IEEE Transactions on Nuclear Science, p. 2267-2274, ISSN : 1558-1578, DOI : 10.1109/23.45434.

[Zus+13] Loïc ZUSSA et al., "Power supply glitch induced faults on FPGA : An in-depth analysis of the injection mechanism", in : *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, ISSN : 1942-9401, juill. 2013, p. 110-115, DOI : 10.1109/IOLTS.2013.6604060.