COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université de Bretagne Occidentale

HABILITATION À DIRIGER DES RECHERCHES DE

L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE

ÉCOLE DOCTORALE N° 644
Mathématiques et Sciences et Technologies de l'Information et de la Communication en Bretagne Océane
Spécialité : *Informatique*

Par
Pascal COTRET

Contributions à la sécurité à la frontière logicielle - matérielle

Habilitation à Diriger des Recherches présentée et soutenue à l'Université de Bretagne Occidentale, le 27 novembre 2025
Unité de recherche : Lab-STICC

Rapporteurs avant soutenance :

Aurélien FRANCILLON	Professeur, Eurecom (Sophia)
Régis LEVEUGLE	Professeur, TIMA (Grenoble)
Jean-Christophe PREVOTÉT	Professeur, INSA (Rennes)

Composition du Jury :

Président·e :	David ESPES	Professeur, Lab-STICC (Brest)
Examinateur·trice·s :	Lilian BOSSUET	Professeur, Hubert-Curien (Saint-Étienne)
	Steven DERRIEN	Professeur, Lab-STICC (Brest)
	Guy GOGNIAT	Professeur, Lab-STICC (Lorient)

UNIVERSITÉ DE BRETAGNE OCCIDENTALE

ÉCOLE DOCTORALE MATHSTIC

Mathématiques et Sciences et Technologies de l'Information et de la Communication

Habilitation à diriger des recherches

de l'Université de Bretagne Occidentale

Présentée et soutenue par

Pascal Cotret

Contributions à la sécurité à la frontière logicielle - matérielle

soutenue le 27 novembre 2025

Jury :

Rapporteurs :	Aurélien Francillon	EURECOM, Sophia Antipolis
	Régis Leveugle	TIMA, Grenoble
	Jean-Christophe Prévotet	IETR, Rennes
Examinateurs :	Lilian Bossuet	Hubert-Curien, St-Étienne
	Steven Derrien	Lab-STICC, Brest
	David Espes	Lab-STICC, Brest
	Guy Gogniat	Lab-STICC, Lorient

Table des matières

Table des figures

Liste des tableaux

Liste des listings

Chapitre 1

Introduction

Sommaire

1.1 Introduction

Ce mémoire d'habilitation à diriger des recherches porte sur mon activité d'enseignant-chercheur qui a eu lieu à CentraleSupélec sur le campus de Rennes entre 2014 et 2017 et à l'ENSTA sur le campus de Brest depuis la rentrée scolaire 2019. À la suite de mon doctorat obtenu en décembre 2012 à l'Université de Bretagne-Sud, j'ai effectué un séjour d'ATER à l'ENSSAT Lannion puis un séjour postdoctoral au CEA LIST à Saclay.

La Figure 1.1 présente une chronologie des différents projets de recherche et des différentes thèses encadrées depuis mon arrivée à CentraleSupélec à la rentrée 2014. Entre mes expériences à CentraleSupélec et l'ENSTA, j'ai travaillé chez Thales sur un poste de développeur embarqué, mais j'ai pu maintenir une activité de recherche sur mon temps libre pour continuer l'encadrement des thèses initiées à CentraleSupélec. Des informations complémentaires sont disponibles en Annexe A.

Mes premières activités de recherche se focalisaient sur un problème de protection des architectures reconfigurables FPGA : comment peut-on proposer une solution flexible et performante pour des attaques visant les mémoires d'un système embarqué ? Une solution a été proposée avec un algorithme de chiffrement et des mécanismes matériels de protection des communications internes à l'architecture implémentée sur le circuit qui apportaient une certaine flexibilité sur les propriétés de confidentialité et d'intégrité ainsi qu'un système de filtres au niveau du bus de communication pour une protection en quasi-temps réel avec une capacité de reconfiguration des règles de sécurité.

Avec l'avènement des architectures dites hétérogènes comprenant un processeur "en dur" (le plus souvent basé sur l'architecture ARM) et un circuit reconfigurable de type FPGA, mes activités de recherche se sont dirigées vers des

problématiques où le matériel et le logiciel peuvent être associés pour proposer des solutions de sécurité originales.

Plus récemment, l'architecture de jeu d'instructions *open-source* RISC-V a apporté un nouvel élan à la sécurité embarquée grâce à sa modularité qui permet de proposer des jeux d'instructions complémentaires et des implémentations de processeurs sécurisés à différents niveaux. Mes travaux de recherches s'articulent sont désormais majoritairement autour de ce type d'architectures.

1.2 Structure du manuscrit

Le manuscrit présente dans un premier temps une synthèse des différentes contributions qui ont été abordées dans les projets visibles dans la Figure 1.1. La suite du manuscrit n'est pas organisée de manière chronologique, mais plutôt thématique. Mes différents travaux de recherche traitent majoritairement de cybersécurité avec l'utilisation et l'étude des architectures reconfigurables FPGA comme point commun dans un contexte "hybride" où le logiciel et le matériel ont tous les deux leur importance. L'articulation des chapitres principaux de ce manuscrit suit une approche descendante de la cybersécurité des systèmes embarqués :

- Le Chapitre 2 décrit deux contributions où la sécurité est à la frontière entre le logiciel et le matériel. Les travaux présentés s'intéressent à des architectures complexes qui peuvent prendre deux formes : 1) un processeur généraliste associé à un coprocesseur dédié à garantir la sécurité d'un code qui s'exécute sur un processeur généraliste non modifiable ; 2) une modification du pipeline d'un processeur pour sécuriser du code compilé à la volée dans un contexte utilisant des machines virtuelles embarquées.
- Le Chapitre 3 présente quant à lui les contributions dans lesquelles on développe des extensions matérielles à un niveau d'abstraction plus élevé. Dans la Section 3.1, la sécurité est étudiée au niveau de l'architecture du système sur puce. La Section 3.2 s'intéresse à la façon dont on peut optimiser l'implémentation d'algorithmes d'intelligence artificielle sur des composants FPGA pour réaliser de la détection d'intrusions dans un contexte de cyberdéfense navale.
- Le Chapitre 4 présente les contributions où la sécurité se situe au niveau de la microarchitecture du système embarqué, ce qui est le cas des deux thèses mentionnées dans cette partie où on s'intéresse principalement aux fuites ayant lieu dans les mémoires cache des processeurs. Là aussi, la sécurité aura un impact sur la couche logicielle (compilateur, services de sécurité) et sur la couche matérielle (extensions nécessaires pour le bon fonctionnement du logiciel renforcé).

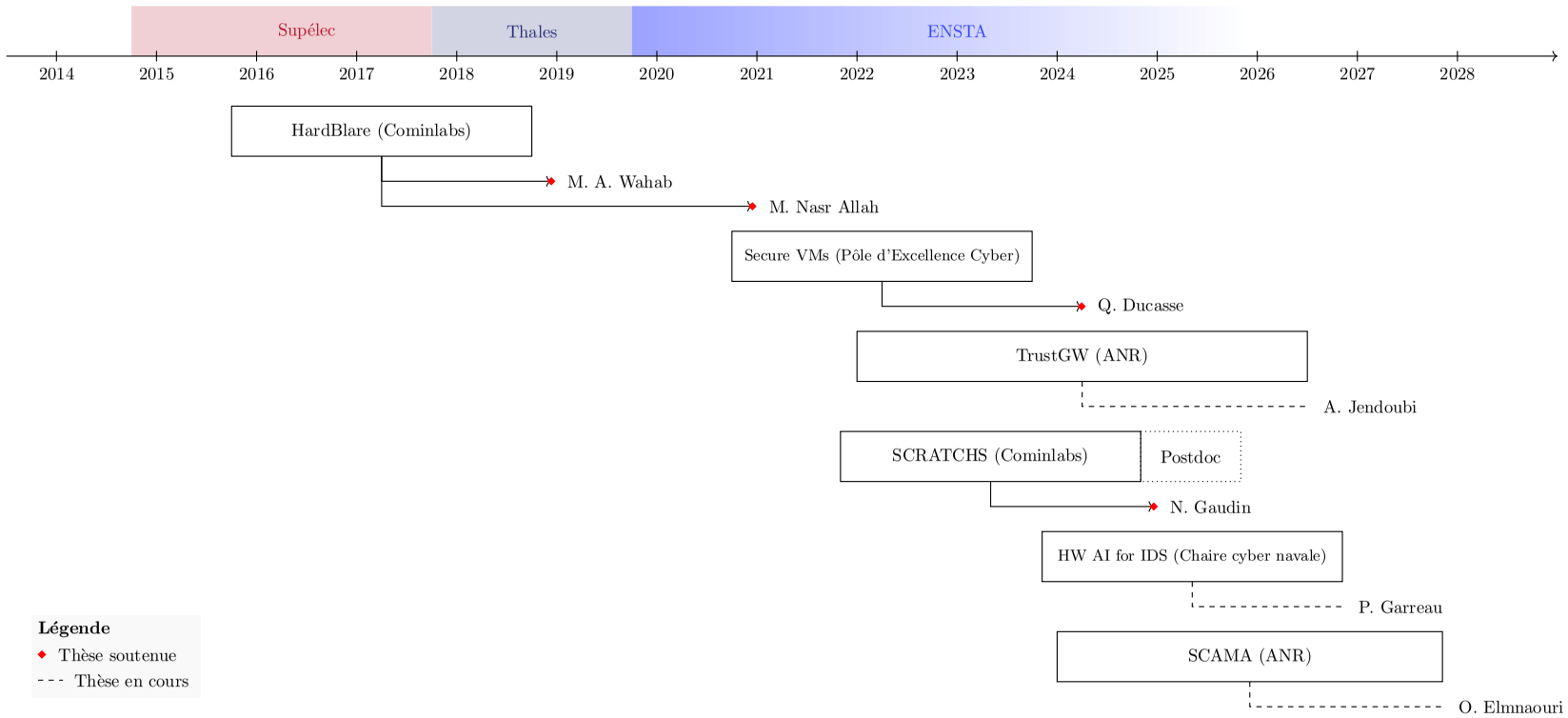


FIGURE 1.1 – Chronologie des thèses encadrées.

Chapitre 2

Contributions à la sécurité hybride logiciel - matériel

Sommaire

2.1 HardBlare

La Section 2.1 présente une synthèse du projet HardBlare dans lequel nous avons proposé une approche qui permet de détecter des attaques bas-niveau dans un logiciel s'exécutant sur un processeur ARM avec une solution qui combine un composant matériel FPGA et des modifications dans la couche logicielle.

2.1.1 Introduction au Dynamic Information Flow Tracking (DIFT)

Le DIFT est une technique qui se décompose en deux étapes :

- La "coloration" des données par des métadonnées appelées *tags* (on parlera aussi dans la suite de l'action de "teinter" des données) et une politique de sécurité définissant la relation entre ces tags.

- La propagation des tags pendant l'exécution du programme ainsi que la détection des violations de politique de sécurité.

Le DIFT peut être réalisé à différents niveaux qu'on peut identifier dans la Figure 2.1.

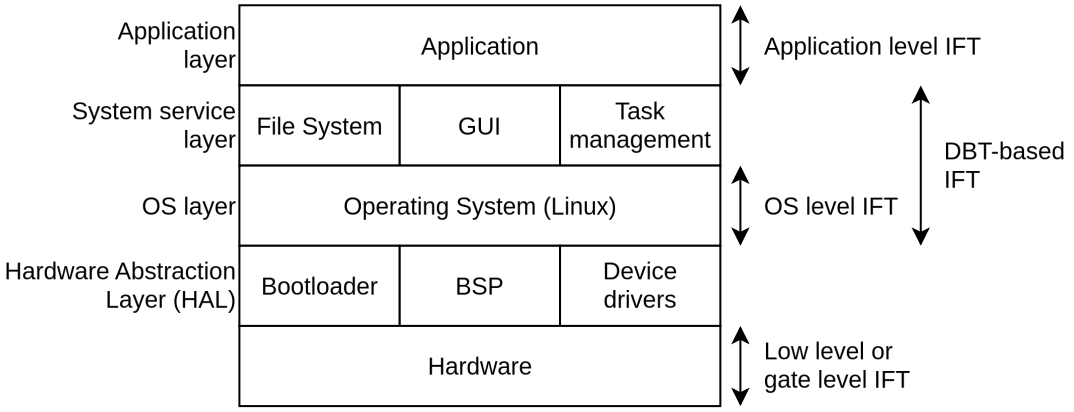


FIGURE 2.1 – Architecture logicielle d'un système embarqué.

L'Information Flow Tracking (IFT) au niveau applicatif (*application-level IFT*) contrôle les flux d'informations entre les variables. L'initialisation, le suivi et le contrôle des tags sont réalisés dans un compilateur modifié ou en analysant le programme. La Figure 2.2 présente un exemple de code C et le code de l'IFT associé.

```c
char buffer;
int a,b,c;
c = a+b;
print(c);
```

```c
char tag_a=1,tag_b=0,tag_c=0;
tag_c=tag_a|tag_b;
if(tag_c==TAG_PRIVATE)
    print("Secure information being leaked");
```

FIGURE 2.2 – Code C et code IFT associé.

Un des problèmes de cette approche est le fait qu'un tag soit associé à chaque variable du programme, ce qui entraîne un surcoût important à l'exécution.

Pour palier à cela, il existe aussi un IFT au niveau du système d'exploitation ou Operating System (OS) (*OS-level IFT*) : dans ce cas, on teinte les fichiers qui sont manipulés par l'application. Cela réduit la quantité de tags, mais augmente également la quantité de faux positifs. Parmi les solutions existantes, Histar [1] est un OS qui a été conçu pour fournir des politiques de sécurité. D'autres approches telles que Blare [2] modifient l'OS : plus précisément, Blare fournit un moniteur de flux d'information qu'on peut intégrer dans un noyau Linux qui va également gérer la lecture et l'écriture des tags.

Dans la Figure 2.1, il existe encore d'autres approches d'IFT :

- Le *gate-level IFT* se fait au moment de la création du circuit en ajoutant de la logique matérielle.
- Des environnements de Dynamic Binary Translation (DBT) peuvent être utilisés pour faire du contrôle de flux d'information [3], [4]. Malheureusement, ces frameworks engendrent des surcoûts en temps d'exécution trop importants.

Le *low-level IFT* va teinter les registres et les adresses mémoire pour développer des accélérateurs matériels qui vont diminuer le surcoût lié à l'IFT. C'est l'approche utilisée dans HardBlare et qui va être détaillée dans la suite de la Section 2.1.

2.1.2 Approches logicielles-matérielles pour le DIFT

Dans l'article sur la solution Raksha [5], Dalton et al. montrent que les solutions de DIFT logicielles peuvent dégrader le temps d'exécution d'un facteur 37. Par conséquent, il y a un intérêt naturel à étudier les solutions matérielles qui permettraient d'accélérer ces traitements. La solution la plus directe est de développer un processeur dédié au DIFT comme cela a été fait dans le projet SAFE [6] : malheureusement, il implique d'avoir une chaîne logicielle spécifique à ce nouveau composant. D'autres approches proposent de modifier des processeurs existants et d'y ajouter des fonctions de DIFT, on en distingue trois catégories.

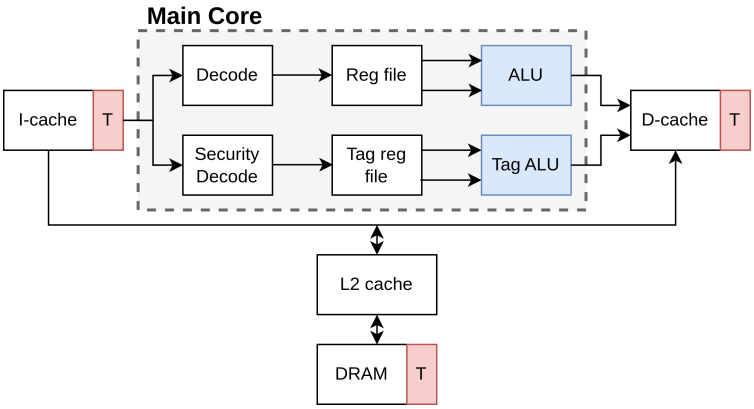


FIGURE 2.3 – Architecture *in-core* (figure adaptée de [7]).

L'approche *in-core* suggère de modifier en profondeur le pipeline du processeur, c'est le cas dans la solution Raksha [5] qui est présentée dans la Figure 2.3 et qui est basée sur l'architecture SPARC. Des tags sont rajoutés au niveau des mémoires ainsi qu'aux différents étages du processeur pour traiter en parallèle la donnée ainsi que le tag associé : par exemple, si la donnée est en cours de

traitement par l'Arithmetic Logic Unit (ALU), un module traite les tags des données. Le processeur est modifié de manière assez conséquente. De plus, étant donné que la solution proposée dans le projet HardBlare visait une architecture à base de processeur ARM, cette approche n'était pas utilisable dans le contexte du projet HardBlare.

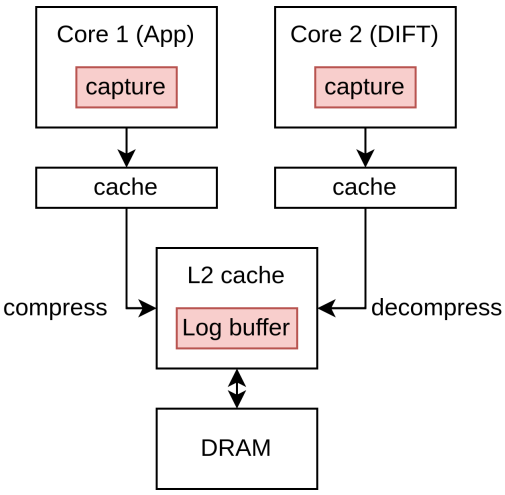


FIGURE 2.4 – Architecture off-loading (figure adaptée de [7]).

La deuxième approche, dite *off-loading*, sépare le traitement DIFT de l'exécution du programme principal en utilisant un deuxième processeur généraliste (voir Figure 2.4). Les données nécessaires au DIFT sont stockées dans une mémoire partagée (un buffer dans le cache L2 dans la Figure 2.4).

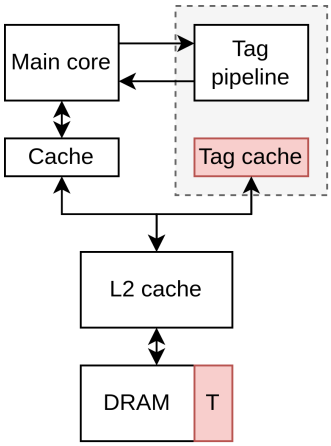


FIGURE 2.5 – Architecture off-core (figure adaptée de [7]).

D'autres solutions, dites *off-core*, ont implémenté des coprocesseurs dédiés qui peuvent se greffer à un processeur existant (voir Figure 2.5). L'équipe derrière Raksha [5] a proposé en 2009 une évolution de leur solution [7]. L'enjeu

principal de cette solution est l'interface entre le processeur principal et le co-processeur DIFT : envoyer les informations nécessaires au DIFT requiert de l'instrumentation et de l'analyse statique et amène un surcoût temporel de 90% [8].

Deng et al. [9], [10] ont proposé un accélérateur matériel qui peut s'exécuter jusqu'à 1 GHz ; cependant, il nécessite de nombreuses informations du processeur principal qui le rend incompatible avec un processeur ARM.

Lee et al. [11] proposent l'idée d'utiliser la Core Debug Interface (CDI) présente dans leur processeur softcore pour récupérer les informations nécessaires au DIFT. Il est possible de récupérer les mêmes informations avec le composant de debug ARM CoreSight Event Trace Macrocell (ETM). Néanmoins, le SoC Zynq utilisé dans HardBlare n'utilise pas l'ETM mais le Program Trace Macrocell (PTM) qui n'envoie les informations que pour les branchements ou les interruptions : par conséquent, l'approche présentée dans [11] n'était pas encore complètement compatible.

2.1.3 HardBlare : une approche DIFT pour ARM

Le Tableau 2.1 présente une comparaison de certaines caractéristiques de travaux existants. Aucun travail ne s'est appliqué à une architecture ARM. Les seuls travaux pouvant être compatibles avec un processeur hardcore ([8], [11]) ont été implémenté sur des processeurs *softcore*.

TABLE 2.1 – Comparaison avec des travaux existants.

Article	Type d'approche	Type de CPU	Cible expérimentale	Portabilité hardcore	Interface communication	Interface simulée	Coprocesseur isolé
[5] [12]	In-core	Softcore	Leon3	Non	Signals	N/A	Non
[13] [14]	Off-loading	Softcore	Leon3	Non	Log buffer en cache	Non	Non
[7] [9], [10]	Off-core	Softcore	Leon3	Non	Signals	Non	Non
[8] [15]	Off-core	Softcore	Leon3	Oui	Bus système	Non	Non
[11]	Off-core	Softcore	Leon3	Oui	CDI	Oui	Non
ARMHEx	**Off-core**	**Hardcore**	**ARM Cortex-A9**	**Oui**	**EMIO et Bus système**	**Non**	**Oui**

La solution ARMHEx développé dans le cadre du projet HardBlare nécessite d'extraire des informations s'exécutant sur le processeur ARM afin de pouvoir effectuer le travail d'analyse dans le composant FPGA.

Le listing 2.1 est un exemple pour illustrer ce qui va être utilisé pour réaliser le DIFT. Ce programme lit deux fichiers (un public et un privé) et écrit le

contenu de l'un des deux dans un troisième fichier selon une condition sur un nombre aléatoire.

```c
int main () {
  int file_public, file_secret, file_output;
  char public_buffer [1024];
  char secret_buffer [1024];
  char *temporary_buffer;
  file_public = open("files/public.txt", O_RDONLY);
  file_secret = open("files/secret.txt", O_RDONLY);
  file_output = open("files/output.txt", O_WRONLY);
  read(file_public, public_buffer, 1024);
  read(file_secret, secret_buffer, 1024);
  srand(time(NULL));
  if((rand()%2)==0){
    temporary_buffer = public_buffer;
  }
  else{
    temporary_buffer = secret_buffer;
  }
  write(file_output, temporary_buffer, 1024);
  return 0;
}
```

Listing 2.1 – Exemple de code C pour DIFT.

Ce code C est compilé en utilisant Low-Level Virtual Machine (LLVM) pour obtenir un binaire pour l'architecture ARM-v7 (architecture du processeur ARM visé dans le projet HardBlare). Un graphe de flot de contrôle simplifié est présenté dans la Figure 2.6. Pour ce code, la trace (autrement dit, les adresses générées par les composants CoreSight) de la fonction `main` est affiché dans le listing 2.2.

```
10618 10494 10634 10494 10648 10494 1065c 10464 10678 10464
 ↪  10690
1047c 10698 10470 106a4 10458 106a8 106c8 10440 106e4 00000
 ↪  00000
```

Listing 2.2 – Trace décodée de la fonction `main` présentée dans le listing 2.1.

Dans le listing 2.2, la première valeur décodée (`0x10618`) correspond à l'adresse de départ du premier bloc de base visible dans la Figure 2.6. La valeur suivante (`0x10494`) est la valeur de l'appel de la fonction `call`. Ensuite, l'adresse de retour de la fonction `call` (`0x10630+4`). Cet exemple montre que la trace décidée permet de déterminer quel bloc de base est en cours d'exécution. Comme la trace est générée uniquement pour les instructions qui changement le flot du programme (telles que des branchements), toutes les autres instructions ne sont pas tracées : par exemple, les instructions entre les adresses `0x10618` et `0x10630` ne génèrent pas de trace.

Une analyse statique est effectuée hors-ligne pour déterminer les flots d'informations à l'intérieur de chaque bloc de base. Cependant, l'analyse statique ne permet pas de récupérer toutes les informations : les adresses mémoire utilisées dans les `ldr` et les `str` ne peuvent pas être récupérées statiquement.

Pour récupérer ces adresses, le binaire est instrumenté pendant la compilation avec une passe LLVM. De plus, l'application s'appuie sur des bibliothèques externes pour tirer profit des appels système et effectuer des opérations spécifiques. Par conséquent, on devrait également récupérer les flux d'information du code de ces librairies. Quatre étapes permettent de régler ce problème de visibilité :

1. Une trace obtenue par les composants CoreSight présent dans le processeur ARM du SoC Zynq.
2. Une analyse statique pour comprendre les flux d'information à l'intérieur des blocs de base.
3. Une instrumentation pour envoyer les adresses mémoire au coprocesseur.
4. Des First In First Outs (FIFOs) dédiées pour la communication entre l'OS et le coprocesseur DIFT.

La Figure 2.7 présente les composants CoreSight présents dans un Zynq Z-7020 ainsi que le chemin pris par les traces depuis le processeur jusqu'à l'ETB accessible par la zone reconfigurable.

On peut classer les composants CoreSight en quatre catégories :

- Contrôle. Contrôle et accès aux composants CoreSight (par exemple, Cross Trigger Interface (CTI), Cross Trigger Matrix (CTM), Debug Access Port (DAP) et Embedded Cross Trigger (ECT)). Dans le cadre du projet HardBlare, seul le DAP est utilisé dans sa configuration standard : il permet d'avoir des ports Advanced Peripheral Bus (APB) pour communiquer avec plusieurs sources de debug. Il n'est pas présenté dans la Figure 2.7.
- Génération de traces pour les instructions processeur (PTM, Fabric Trace Monitor (FTM) and Instrucmentation Trace Macrocell (ITM)).
- Lien entre les sources et ports de sortie (funnel and replicator).

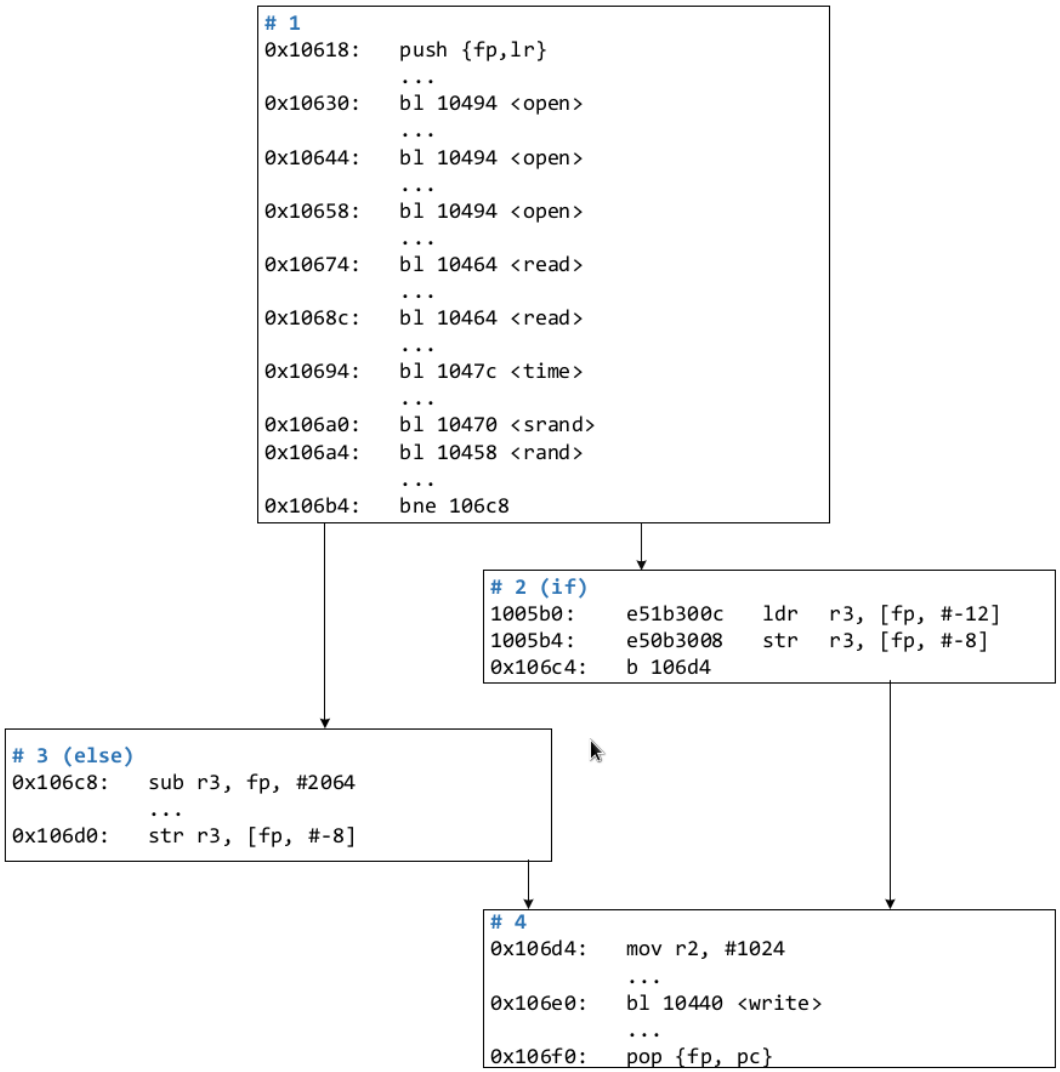


FIGURE 2.6 – CFG simplifié du code C présenté dans le Listing 2.1.

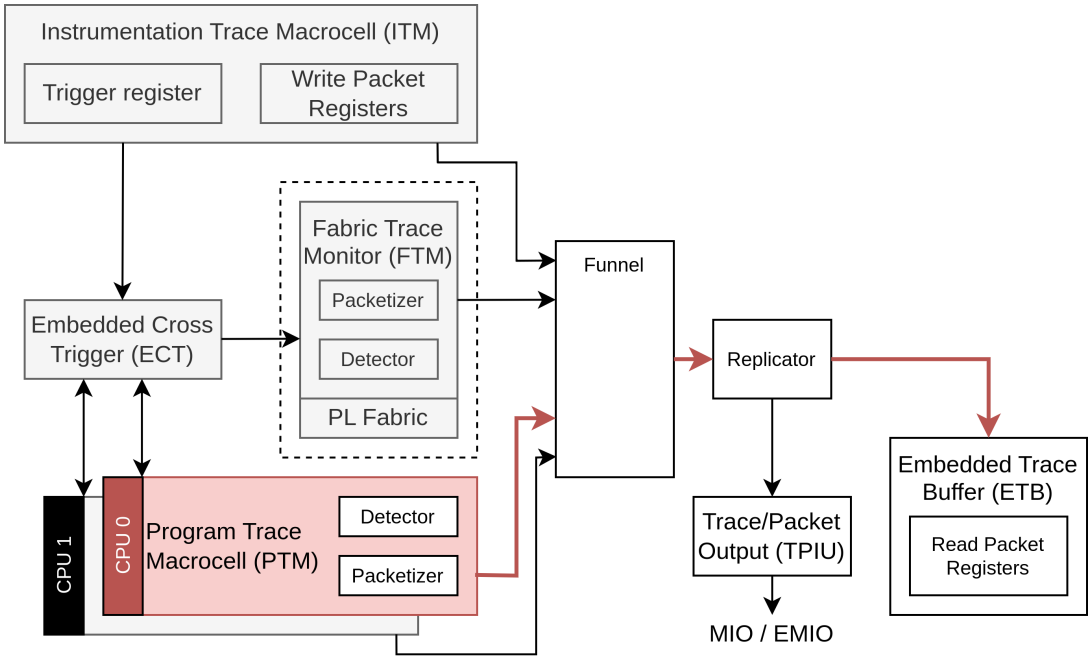


FIGURE 2.7 – Composants CoreSight dans un SoC Zynq. Le chemin en rouge indique le parcours suivi par les traces de debug.

- Stockage ou export de traces (Embedded Trace Buffer (ETB) et Trace Port Interface Unit (TPIU)). La trace peut être transmise à une mémoire interne (l'ETB), le coprocesseur FPGA (par le TPIU) ou même des broches dédiées.

Ensuite, l'analyse statique peut être réalisée de deux façons :

- Après la compilation en utilisant le désassembleur Capstone : une fois le binaire désassemblé, Capstone va générer les annotations pour chaque instruction.
- Pendant la compilation avec le compilateur LLVM. Ce compilateur a l'avantage de donner plus d'informations qu'un désassembleur et est capable de générer les flux d'informations de codes obfusqués.

La trace décodée permet de savoir que bloc de base est en cours d'exécution. L'analyse statique permet de déterminer les flux d'informations dans chaque bloc de base. Néanmoins, certaines informations ne peuvent pas être déterminées statiquement. Par exemple, si une instruction de chargement `ldr` est analysée statiquement, l'adresse mémoire ne peut pas être calculée. Par conséquent, les adresses mémoire manquantes pour des `ldr` et `str` sont récupérées avec une instrumentation. Une IP a été développée dans le coprocesseur présent dans la zone reconfigurable à cet effet.

L'architecture globale de la solution est présentée dans la Figure 2.8. On y trouve : le Program Flow Trace (PFT) decoder qui décode les traces, le ARMHEx coprocessor qui est le coeur de DIFT à proprement parler et le Tag Register File (TRF).

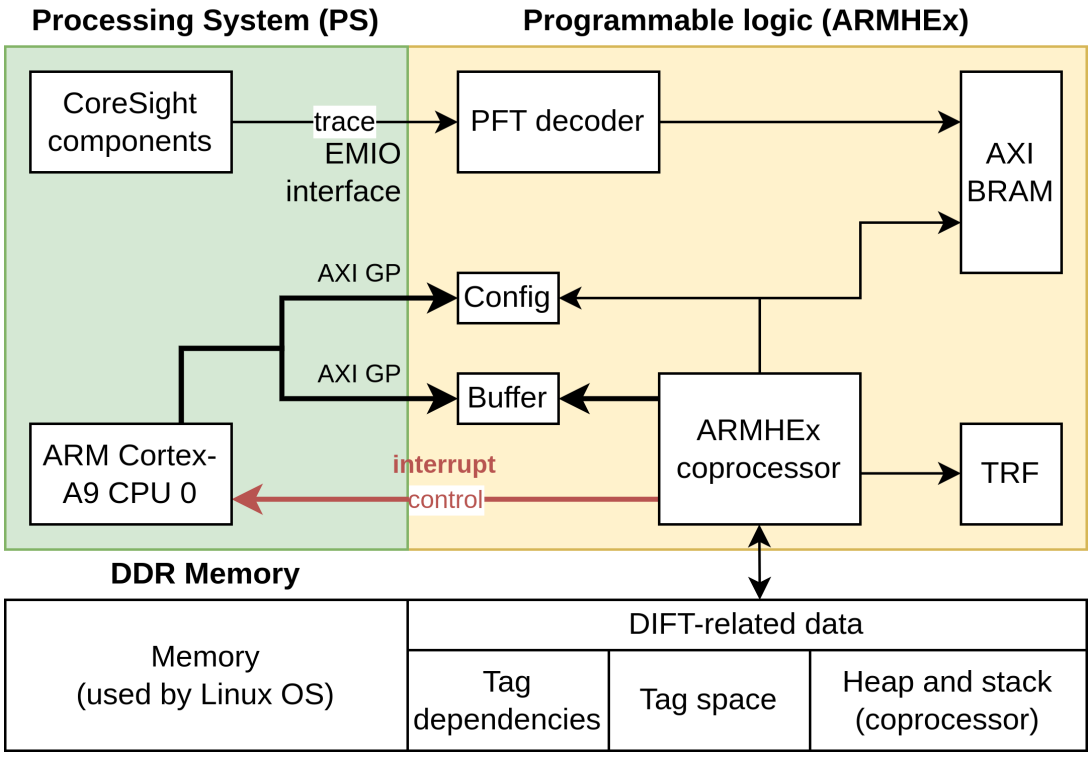


FIGURE 2.8 – Architecture interne de ARMHex avec le processeur ARM dans la partie PS et le coprocesseur DIFT ARMHEx dans la partie PL.

2.1.4 Évaluation et résultats

La Table 2.2 montre une comparaison entre ARMHEx et les autres approches *off-core* présentées précédemment. Contrairement aux autres travaux, ARMHEx est construit autour d'un processeur ARM hardcore : cela ouvre des perspectives intéressantes étant donné que ce travail pourra être plus facilement porté sur des systèmes embarqués existants. Pour ce qui est de la surface, ARMHEx a le meilleur ratio coprocesseur/processeur, car le processeur utilisé est un Cortex-A9 qui a environ 26 millions de portes [16]. De plus, le surcoût en communication est 10 fois plus faible que dans [8].

TABLE 2.2 – Comparaison avec des approches existantes.

Approches	Kannan [7]	Deng [9]	Heo [8]	ARMHEx
Portabilité hardcore	Non	Non	Oui	Oui
Processeur	Softcore	Softcore	Softcore	Hardcore
Surcoût communication	N/A	N/A	60%	5,4%
Surcoût surface	6,4%	14,8%	14,47%	0,47%
Surface (portes)	N/A	N/A	256177	128496
Surcoût consommation	N/A	6,3%	24%	8,45%
Fréquence maximale	N/A	256 MHz	N/A	250 MHz

2.2 Protection de machines virtuelles embarquées pour RISC-V

2.2.1 Introduction sur les VMs et leur sécurité

La sécurité des données dans un système embarqué peut être implémentée à différents niveaux : dans l'architecture matérielle, dans la couche logicielle (soit au niveau du système d'exploitation (OS) ou sur le programme qui sera exécuté) ou alors grâce à des environnements d'exécution spécifiques.

Les machines virtuelles ou Virtual Machines (VMs) dédiées à l'exécution de code sont une de ces solutions. Il en existe différentes catégories dont on peut retrouver une description détaillée dans l'ouvrage de Smith et al. [17]. Elles sont en capacité d'exécuter du code en faisant abstraction des phases de compilation et d'allocation mémoire : on en retrouve, par exemple, dans Java et Python ou encore dans les navigateurs web dès lors qu'on veut exécuter du code JavaScript (par exemple, SpiderMonkey dans Firefox).

La structure générale d'une machine virtuelle est présentée dans la Figure 2.9. On y retrouve principalement quatre composants :

- Le compilateur de bytecode (*bytecode compiler*). Ce compilateur transforme le code source représenté sous forme d'Abstract Syntax Tree (AST) en une représentation intermédiaire spécifique à la VM visée : on parle alors de *bytecode*.
- L'interpréteur (*interpreter*) va quant à lui, décoder et analyser ce bytecode pour exécuter les actions correspondantes.
- Le compilateur Just-in-Time (JIT) va permettre de recompiler pendant l'exécution une section de bytecode qui peut être vu comme étant "utile", car appelée fréquemment. Cette partie de code est elle-même recompilée

en une représentation intermédiaire : là où le bytecode est conçu pour être utilisé par l'interpréteur, la représentation issue du JIT est optimisée pour le code machine.

- Enfin, le ramasse-miette (ou *garbage collector*) est quant à lui responsable des allocations mémoires pour la machine virtuelle.

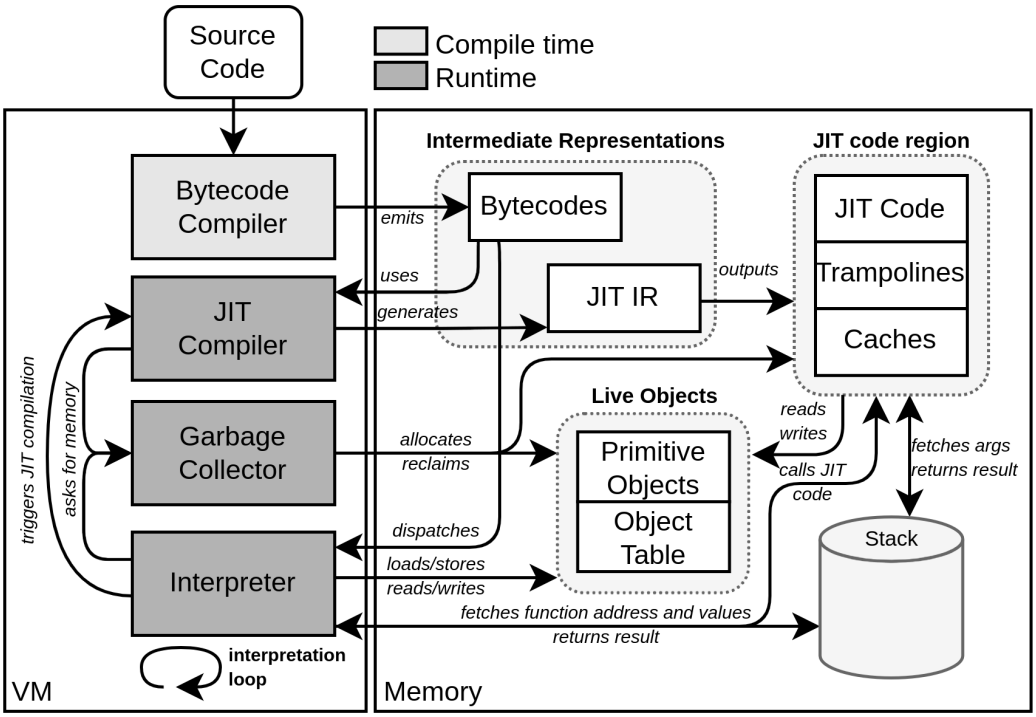


FIGURE 2.9 – Schéma synthétique d'une machine virtuelle. Figure adaptée de [18].

Dans ces VMs, le module responsable de la compilation à la volée est le compilateur JIT. Les attaques sur les machines virtuelles sont basées sur des attaques sur du code compilé statiquement et qui tirent profit des opérations bas-niveau réalisées par le JIT. Ces attaques peuvent être divisées en trois catégories :

- Les attaques par injection de code. Ces attaques, définies dans un article du magazine Phrack [19] montrent qu'il est possible d'attaquer le système en manipulant convenablement la pile ou le tas avec des *shellcodes*.
- Les attaques dites *code-reuse*. Ces attaques sont plus connues sous la forme de Return-Oriented Programming (ROP) comme cela a pu être défini dans un article de Hovav Shacham en 2007 [20] : elles sont généralement présentes sous la forme de gadgets, des extraits de codes qui finissent par une instruction de type `ret` qui permet de pointer sur une adresse mémoire qui contiendrait un code malveillant à exécuter.

- Les attaques dites *data-only*. Chen et al. parlent de *non-control-data attacks* [21] qui s'opposent aux *control-data attacks* où le but de l'attaque est de modifier les adresses retour ou les pointeurs de fonction pour exécuter du code malveillant. Les attaques *data-only* visent les données sensibles d'un programme (données de configuration, identification d'un utilisateur...)

Toutes ces attaques ont pu être appliquées dans des contextes avec des machines virtuelles [18], [22], [23]. Différents mécanismes de protection ont été proposés, on peut en extraire trois axes principaux :

- Diversification. Comme l'explique Lian et al. [24], ces mécanismes peuvent se matérialiser sous la forme d'intra-instructions ou de randomisation de la disposition du code : par exemple, par les registres ou en insérant un nombre aléatoire d'instructions `nop` entre des instructions légitimes.
- Protection mémoire. Pour se prémunir des exécutions de code dans la pile, il est possible de définir des zones mémoire W⊕X [25] ou d'implémenter une Data Execution Prevention (DEP) [26]. Cependant, le concept de W⊕X est contradictoire avec l'utilisation d'un JIT qui doit réécrire en permanence le code machine.
- Restriction des capacités qui consiste à isoler les codes n'étant pas de confiance. On y trouve par exemple des solutions de Control-Flow Integrity (CFI) où la politique de sécurité définit un CFG que le programme doit suivre [27].
- Isolation assistée par le matériel. C'est dans cette catégorie qu'on trouve les TEEs (par exemple, ARM TrustZone [28], Intel SGX [29] ou Keystone pour RISC-V [30]) ou les clés de protection mémoire (Memory Protection Keys (MPKs) [31]). Ces approches permettent, entre autres, d'isoler du code sensible du reste de l'application.

Les différentes attaques et défenses que nous avons étudiées pendant notre état de l'art ont été proposées pour différentes architectures telles que ARM, x86 ou RISC-V pour les publications les plus récentes. Dans le cadre des travaux de thèse de Quentin Ducasse, nous souhaitions faire des modifications à bas niveau tant sur le logiciel que sur le matériel sur lequel est exécuté la VM. Du fait de la modularité de son jeu d'instruction et la possibilité de pouvoir l'étendre tout en conservant une rétrocompatibilité, RISC-V nous parait le choix le plus opportun pour développer des extensions de sécurité pour des machines virtuelles.

La VM Pharo est une machine virtuelle activement maintenue et utilisée en production. Elle a été portée sur ARMv8 [32]. Nous avons décidé d'étendre son compilateur JIT, Cogit, pour prototyper l'ajout d'instructions JIT dans l'avenir. Dans le cadre de la thèse de Quentin Ducasse, il y a eu une phase de développement significative pour adapter Pharo sur RISC-V : ce travail n'est pas

décrit dans ce manuscrit, mais a été essentiel pour la poursuite des travaux sur ce sujet. De ces différentes attaques et protection, nous avons voulu répondre à deux questions : ① Comment comparer de manière simple ces différentes solutions ? ② Comment avoir une protection à grain fin (au niveau instruction) avec un surcoût limité ?

2.2.2 Gigue : un générateur de logiciels

2.2.2.1 Description du fonctionnement

Gigue [33] est un générateur de binaires qui produits des exécutables en bare-metal qui représente une région de code jitté avec des méthodes remplies de manière aléatoire. Il s'agit d'un outil très paramétrable qui permet de simuler différentes complexités de codes jittés pour accélérer le prototypage d'extensions matérielles comme cela sera le cas dans la Section 2.2.3.

Lorsqu'on propose une extension matérielle pour un processeur RISC-V, des instructions peuvent être rajoutées. Dans un contexte d'exécution avec une machine virtuelle, cela voudrait dire qu'il faudrait modifier le système d'exploitation et le compilateur JIT (symboles en cercle dans la Figure 2.10).

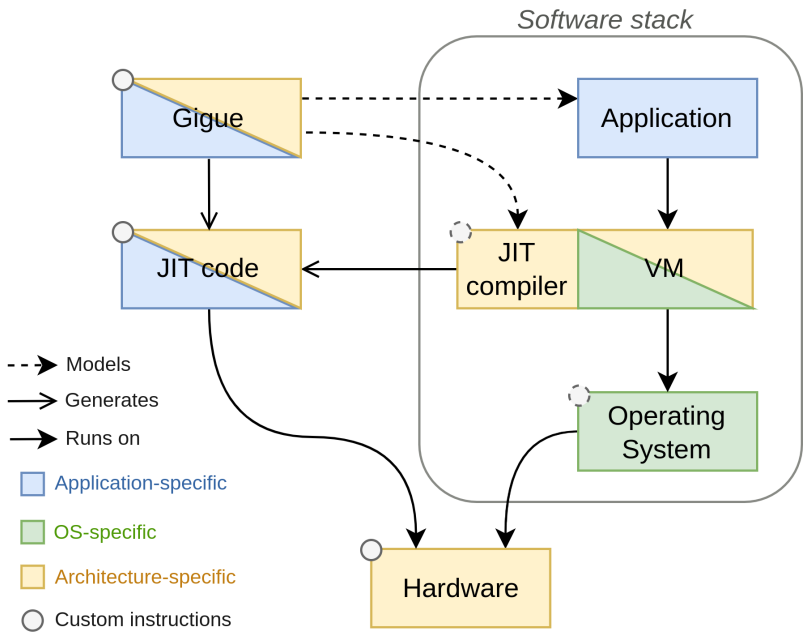


FIGURE 2.10 – Développement de code *jitté* pour un processeur embarqué. Le symbole rond représente les instructions additionnelles qui seraient rajoutées pour une extension de sécurité.

Le code de Gigue est disponible dans un dépôt Github [1] : on peut y paramétrer plusieurs éléments tels que la taille de la région de code jitté, le nombre de registres utilisables, le nombre total de méthodes (ainsi que leur taille individuelle) ou encore le nombre d'appels de fonctions.

Pour le développement de nos extensions matérielles, nous avons étudié deux processeurs RISC-V : Rocket [2] et CVA6 [3]. Nous avons donc dû définir un environnement de test pour les binaires générés par Gigue, celui-ci est présenté dans la Figure 2.11.

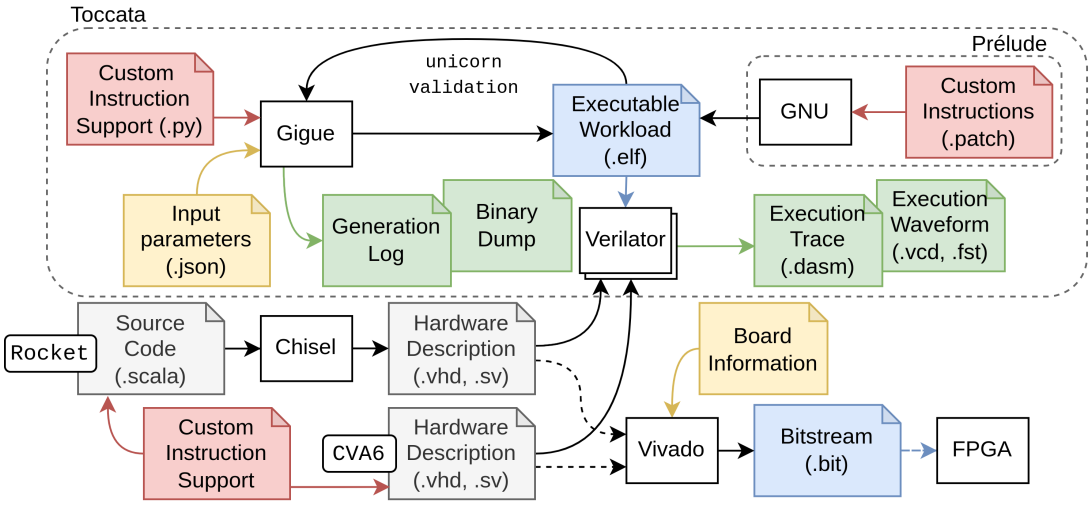


FIGURE 2.11 – Environnement de test pour Gigue à destination des processeurs RISC-V Rocket et CVA6.

Les binaires générés avec Gigue sont exécutés sur des modèles Verilator des deux processeurs grâce à un framework nommé Toccata [4]. Toccata va générer différents fichiers (en vert dans la Figure 2.11) qui permettent d'avoir des traces de l'exécution des différents binaires (dump, log et fichiers `vcd` pouvant être visualisés avec GTKWave).

2.2.2.2 Résultats

En utilisant Gigue, nous avons essentiellement analysé deux paramètres : tout d'abord, le nombre d'appels effectués dans une méthode (taux d'occupation de 1 à 6%) ; puis, le nombre d'accès mémoire (entre 4 et 12% des instructions seront des `load` dans les méthodes). De plus, nous faisons varier le nombre de

1. https://github.com/qducasse/gigue
2. https://github.com/chipsalliance/rocket-chip
3. https://github.com/openhwgroup/cva6
4. https://github.com/QDucasse/gigue/tree/main/toccata

méthodes ainsi que leurs tailles afin d'avoir un échantillon suffisamment représentatif d'applications. Pour le nombre d'appels, à l'initialisation, les méthodes ont un taux d'occupation de 3% et on le fait varier de 1 à 6%. Nous avons également expérimenté la variation de la taille des méthodes en fixant la taille globale de la région de code JIT puis en générant des méthodes de 400, 600 et 800 octets.

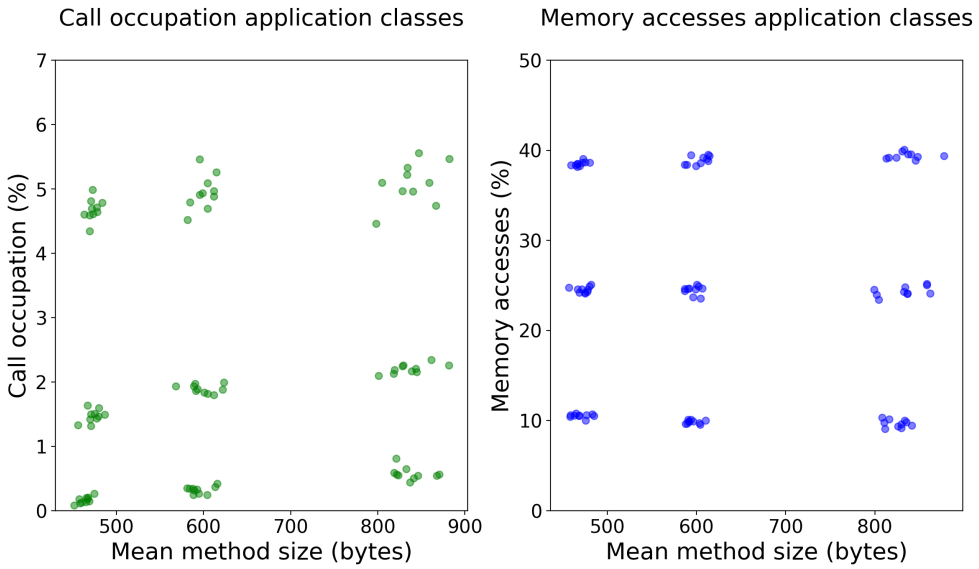


FIGURE 2.12 – Génération des classes d'application.

Les trois axes d'exploration sont présentés dans la Figure 2.12 : dans la figure de gauche, le taux d'appels varie à quantité d'accès mémoire constante ; dans la figure de droite, le nombre d'accès et la taille des méthodes varie à taux d'appels constant. Dans chaque cas, on a 9 "classes d'application" distinctes.

La Figure 2.13 présente les temps d'exécution (en nombre de cycles) des 18 classes d'applications (9 pour la variation d'appels et 9 autres pour les accès mémoire) sur deux processeurs RISC-V : CVA6 (en bleu) et Rocket (en rouge). Nous pouvons observer qu'augmenter le taux d'appels augmente sensiblement la charge sur le coeur. Les méthodes "larges" avec un taux d'appels important génèrent des charges de travail plus longues. Concernant les accès mémoire, leur augmentation engendre également une augmentation du temps d'exécution.

2.2.3 JITDomain : une protection par le matériel

2.2.3.1 Explication

À la suite de cette première contribution, nous avons souhaité tirer profit de la compilation JIT pour proposer JITDomain qui est un framework de sécurité

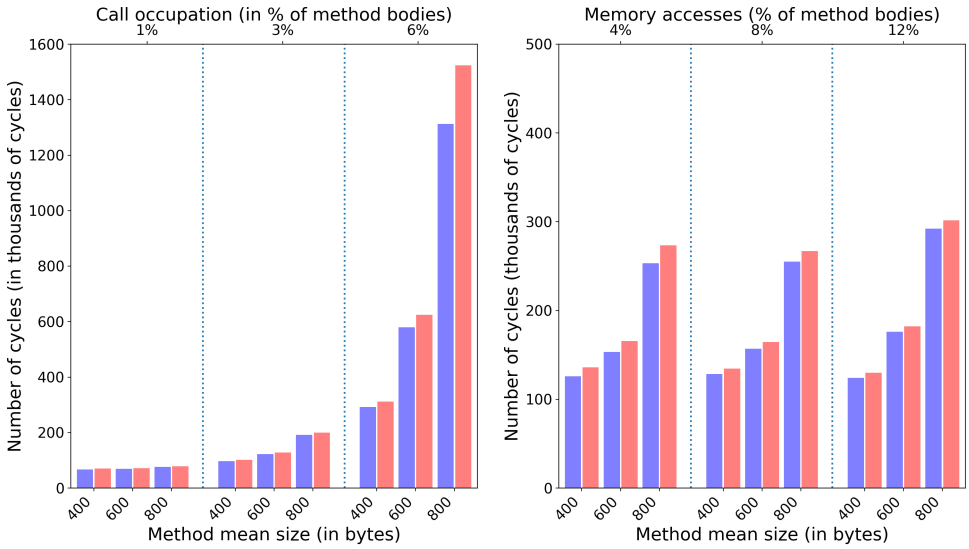


FIGURE 2.13 – Exécution des binaires Gigue sur processeur CVA6 (en bleu) et Rocket (en rouge).

matérielle basé sur le principe d'isolation au niveau instruction qu'on pouvait observer dans des travaux existants [34], [35].

Une isolation de domaine au niveau instruction [34] est établie en dupliquant les instructions d'accès mémoire et en leur donnant des domaines d'exécution autorisés. Le code présent dans un domaine peut accéder uniquement aux données de ce même domaine ; des instructions particulières sont également ajoutées pour pouvoir changer de domaine. Cette isolation est basée sur trois grands principes : **(1)** l'étiquetage des régions mémoire en "domaines" ; **(2)** la duplication des accès mémoire pour le deuxième domaine ; **(3)** l'ajout d'instructions pour changer de domaine.

L'instrumentation de ces domaines dédiés pour la région de code JIT permet d'avoir trois garanties à appliquer au code JIT : le *call stack separation*, le *data access restriction* et le *system call filtering*. Pour arriver à ce but, trois domaines sont définis : le *base domain* où le code de la VM et les données sont stockés dans `basedom`, le *jit domain* qui contient le code JIT et les données associées dans *jitdom* et un *stack domain* qui contient la *shadow stack* des adresses de retour (`stackdom`).

JITDomain a été implémenté sur un processeur CVA6 : il utilise une structure `scoreboard` pour conserver une trace des instructions décodées et de leurs opérandes, ainsi que les potentielles exceptions. Cette structure a été étendue avec deux paramètres principaux : un *target domain*, le domaine auquel l'instruction accède par un accès mémoire ou un changement de domaine ; et un flag *domain change* qui définit si l'instruction doit changer de domaine ou non. En

ce qui concerne l'étiquetage des régions mémoires, plusieurs Control and Status Registers (CSRs) sont ajoutés pour stocker l'information de domaine liée aux régions définies dans le Physical Memory Protection (PMP) : on définit un CSR `dmpcfg` qui contient 16 configurations de domaine (de `dmp0cfg` à `dmp15cfg`), comme on peut le voir dans la Figure 2.14.

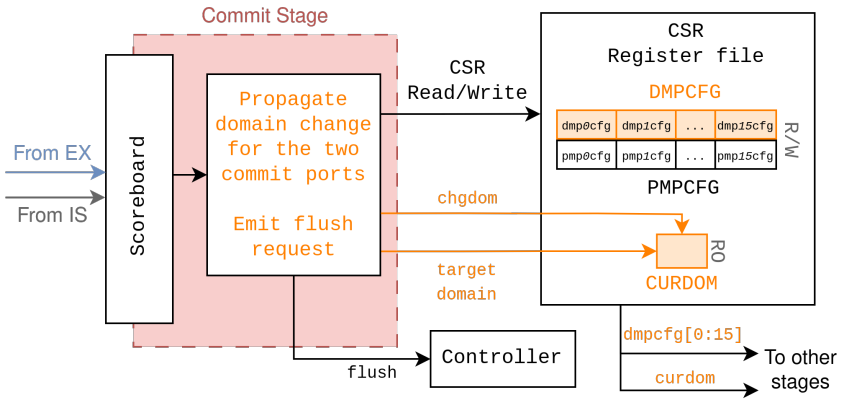


FIGURE 2.14 – Implémentation des CSRs pour la configuration de domaine.

Ensuite, il y a des modifications à apporter pour trois vérifications de domaine.

- Vérification de *code domain*. Dans cette étape, on vérifie le domaine de l'instruction en cours de traitement avec le domain courant présent dans le registre `curdom` : s'il y a une différence entre les deux, une interruption d'instruction illégale est déclenchée. Pour vérifier le *code domain*, il a été nécessaire de rajouter la logique de vérification dans le processeur (le contrôleur d'interruption était déjà, il a juste fallu associer le flag d'instruction illégale déjà présent dans le processeur CVA6 que nous avons utilisé).

- Vérification de *data domain*. Nous avons implémenté un contrôle du domaine de la donnée qui doit être accédée en lecture-écriture sous la forme d'un Domain Memory Protection (DMP) qui se place en amont du PMP. Vérifier le *data domain* a nécessité d'ajouter la logique associée dans le processeur.

- Vérification de *fetch domain*. Une étape similaire est réalisée dans l'étage de *fetch* du processeur : les requêtes et les réponses du cache sont étendues avec des informations de domaine qui sont également vérifiées.

2.2.3.2 Résultats

Pour mesurer l'impact en performance de JITDomain, nous avons utilisé Gigue (décrit dans la Section 2.2.2) pour générer différentes applications. Nous avons utilisé le processeur CVA6 commit `bb80b3f`, Verilator version 5.008 et les binaires ont été compilés avec la version 2.40.0 de la *toolchain* GNU RISC-V.

Une comparaison est faite entre un CVA6 non modifié (CVA6 *baseline*) et la version contenant le mécanisme JITDomain dans la Table 2.3 et la Figure 2.15 pour les différentes applications "call" (variation du taux d'appels dans les fonctions), et la Table 2.4 et la Figure 2.16 pour les différentes applications "memory" (variation du nombre d'accès mémoire et des tailles des méthodes). Les deux figures montrent le surcoût en nombre de cycles (couleur foncée dans les histogrammes) et le Cycles Per Instruction (CPI) (couleur claire).

TABLE 2.3 – Surcoût en performance (nombre de cycles / cycles par instruction).

Taille méthode / *Taux de calls*	**1. 400 octets**	**2. 600 octets**	**3. 800 octets**
1. 1% des instructions	2,43% / 2,34%	1,67% / 1,47%	1,46% / 1,15%
2. 3% des instructions	2,29% / 1,52%	1,56% / 0,72%	0,98% / 0,29%
3. 6% des instructions	1,22% / 0,27%	0,94% / 0,22%	1,01% / 0,49%
Moyenne	**1,51% / 0,95%**		

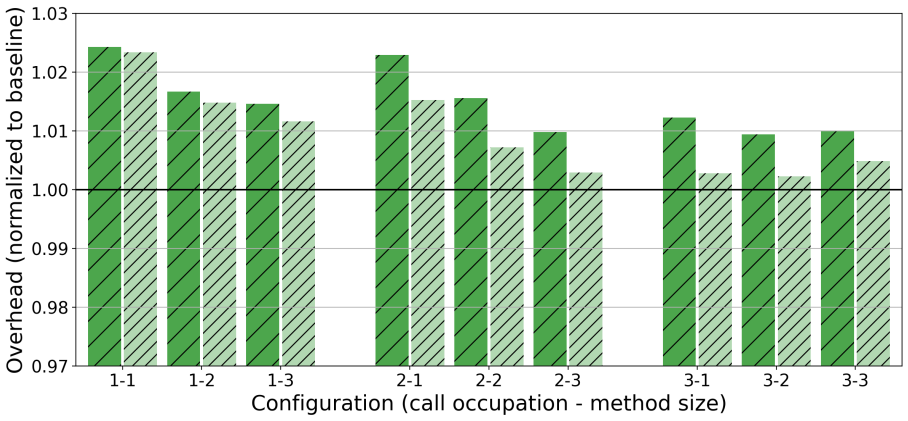


FIGURE 2.15 – Surcoûts en cycles et en CPI (applications *call*).

L'impact en termes de performances sur le nombre de cycles est négligeable : 1,51% pour les appels (*calls*) et 1,27% pour les accès mémoire. Les résultats sont plus visibles lorsque que les méthodes JIT sont de petite taille : 2,4% pour les appels et 2,2% pour les accès mémoire. Le nombre de cycles par instructions est également impacté dans une moindre mesure avec une moyenne de 0,91% pour

les appels et 0,61% pour les accès mémoire. Globalement, on peut constater que l'impact sur les performances est minimal.

TABLE 2.4 – Surcoût en performance (nombre de cycles / cycles par instruction).

Taille méthode / *Accès mémoire*	**1. 400 octets**	**2. 600 octets**	**3. 800 octets**
1. 4% des instructions	2,16% / 1,26%	1,39% / 0,72%	0,86% / 0,31%
2. 12% des instructions	1,69% / 0,96%	1,21% / 0,55%	0,68% / 0,19%
3. 20% des instructions	1,60% / 0,85%	1,20% / 0,50%	0,60% / 0,06%
Moyenne	**1,27% / 0,61%**		

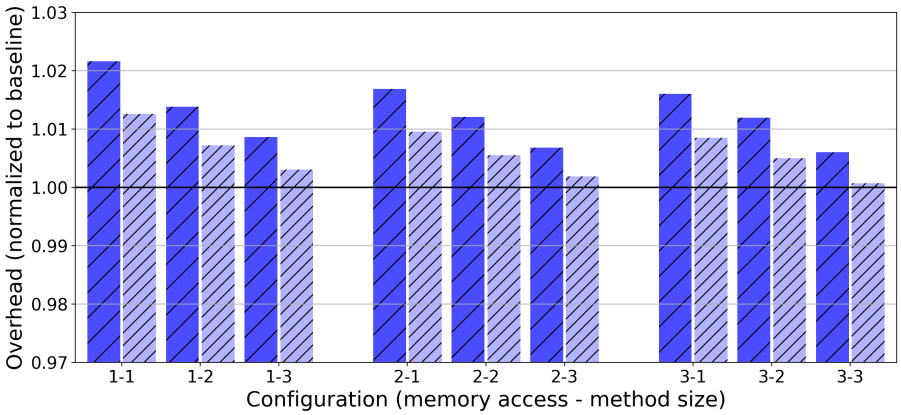


FIGURE 2.16 – Surcoûts en cycles et en CPI (applications *memory*).

Les résultats d'implémentation sur FPGA sont présentés dans la Table 2.5 où on y présente une comparaison entre un processeur CVA6 avec la solution JIT-Domain et un processeur CVA6 non modifié. L'approche JITDomain amène un surcoût inférieur à 0,5% pour les Look-Up Tables (LUTs) et les registres. Nous supposons que cela engendre également un surcoût négligeable sur la consommation globale du circuit. De plus, les modifications matérielles effectuées ne sont pas sur le chemin critique du processeur, la fréquence du circuit reste donc maximale.

	LUT		Slice Registers	
	Baseline	*JITDomain*	*Baseline*	*JITDomain*
Frontend	3190	3171	4058	4060
Decode Stage	609	664	266	269
Issue Stage	14936	15134	8983	9036
Execute Stage	23391	23410	7563	7571
Commit Stage	203	218	0	0
CSR Regfile	2434	2408	1691	1716
Cache Subsystem	6242	6242	2507	2509
Total	**52719**	**52961**	**25482**	**25575**
	(+0,46%)		**(+0,36%)**	

TABLE 2.5 – Résultats d'implémentation de JITDomain pour un processeur RISC-V CVA6 non modifié (*baseline*) et la version avec l'implémentation de JITDomain.

2.3 Conclusion et perspectives

Ce chapitre a présenté les principales contributions des thèses de Muhammad Abdul Wahab et de Mounir Nasr Allah sur le projet Cominlabs HardBlare (Section 2.1) et celle de Quentin Ducasse sur les machines virtuelles embarquées et sécurisées (Section 2.2). Les publications associées à ces deux thèses sont notées dans l'annexe A.2.

DIFT et monitoring logiciel

Les travaux présentés dans la Section 2.1 ont proposé une solution qui permet un suivi (ou *monitoring*) de l'exécution d'un logiciel dans un processeur embarqué ARM en s'aidant des composants de debug présents dans le processeur et en implémentant un co-processeur dans un FPGA.

La thématique autour du DIFT matériel a été poursuivie sous plusieurs formes. On peut citer l'approche Raft [36] qui est un équivalent à l'approche off-core qu'on a développé dans la thèse de Muhammad Abdul Wahab mais où le processeur est désormais basé sur l'Instruction Set Architecture (ISA) RISC-V avec le coprocesseur connecté via une interface dédiée. La solution présentée dans cet article présente malgré tout un surcoût matériel assez important (+34% de LUTs et +93% de registres).

On peut citer également des travaux qui s'intéressent aux attaques en fautes d'une solution de DIFT sur un processeur RISC-V [37].

Sur ces fonctions de *monitoring* de logiciel, il y a encore des éléments à

étudier, notamment autour de l'utilisation des compteurs de performance pour récupérer des métriques sur le comportement du logiciel. Il me semble aussi également intéressant de suivre les évolutions de la spécifications RISC-V autour des mécanismes de trace [38] ce qui pourrait permettre de proposer des solutions encore plus performantes à moindre coût matériel.

Compilation à la volée et RISC-V

Toujours dans une thématique de sécurité combinant des éléments matériels et logiciels, les travaux présentés dans la Section 2.2 ont montré qu'il était possible de sécuriser des machines virtuelles embarquées sur un processeur RISC-V. Plus précisément, les travaux s'articulaient autour de la sécurisation de code JIT par des modifications dans le pipeline du processeur. Les travaux de Quentin Ducasse nous ont permis de publier plusieurs articles dont un article de revue est en cours d'évaluation dans la revue Elsevier "Microprocessors and Microsystems" [39] : cet article se focalise sur l'implémentation matérielle de son approche JITDomain pour séparer l'exécution de logiciels en plusieurs domaines.

Les travaux vont avoir des suites dans le projet SCAMA décrit plus tard dans ce manuscrit dans la Section 4.2 où on souhaite intégrer une compilation à la volée pour protéger des caches contre des attaques par canaux cachés.

Chapitre 3

Contributions aux architectures matérielles sécurisées

Sommaire

3.1 Protection d'une IOMMU

La section décrit une partie des travaux de la thèse d'Aya Jendoubi dont la thèse a lieu dans le cadre du projet ANR TrustGW (https://trustgw.projects.labsticc.fr/). Dans un premier temps, un bref rappel du contexte du projet est effectué avant de présenter une synthèse des travaux en cours.

3.1.1 Contexte du projet

À l'heure de l'Internet des objets, les systèmes embarqués communicants se répandent massivement dans des infrastructures critiques. Malheureusement, ils participent malgré eux à l'augmentation de la surface d'attaque globale des systèmes d'information ce qui représente une menace sans précédent [40]. Il est

donc essentiel de garantir le meilleur niveau de protection pour de tels systèmes qui manipulent des données sensibles. En effet, du fait de leur connectivité, ils font face à de nombreuses menaces.

Dans le cadre du projet TrustGW, le système considéré est composé d'objets connectés à une passerelle (ou *gateway*) qui est elle-même connectée à un ou plusieurs serveurs de calculs. L'architecture de la passerelle qui est au coeur du projet est "hétérogène" logicielle-matérielle et composée de plusieurs processeurs (un processeur bande de base ou BaseBand Processor (BBP) et un processeur générique ou General Purpose Processor (GPP)) et d'accélérateurs matériels implémentés sur FPGA (les ressources FPGA sont virtualisées afin d'avoir une vue uniforme du point de vue des applications). La passerelle embarque plusieurs machines virtuelles afin de pouvoir déployer les services des différents opérateurs qu'elle héberge de façon cloisonnée. Un hyperviseur permet de déployer les machines virtuelles et d'assurer leur sécurité [41]. La Figure 3.1 présente une vue globale de l'architecture prise en compte pour la passerelle avec les différentes ressources logicielles et matérielles qui sont mises en oeuvre dans le projet.

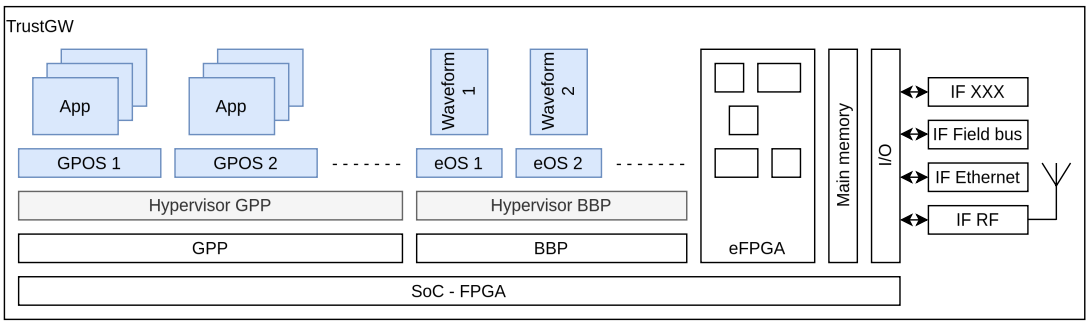


FIGURE 3.1 – Architecture générale de la passerelle.

L'architecture est composée d'un processeur applicatif GPP hébergeant plusieurs machines virtuelles (*GPOS*) déployées par un hyperviseur dédié (*hypervisor GPP*), d'un processeur bande de base (BBP) hébergeant plusieurs machines virtuelles (*eOS*) déployées par un hyperviseur dédié (*hypervisor BBP*) et d'une matrice reconfigurable permettant de déployer des accélérateurs matériels (*eFPGA*). Comme illustré sur la Figure 3.1, une application est déployée en logiciel et matériel créant ainsi un espace d'exécution.

Concrètement, le projet ANR TrustGW se décompose en trois thématiques majeures :

- La sécurisation de l'architecture de communication de la passerelle. Les travaux réalisés par un doctorant au Lab-STICC à l'Université de Bretagne-Sud intègrent notamment l'étude d'architectures de détection d'intrusions dans ce type de passerelle connectée [42].

- La sécurité des applications au sein des machines virtuelles. Ces travaux sont réalisés par un doctorant dans l'équipe INRIA SUSHI [1].
- La sécurisation de l'hyperviseur et des ressources virtualisées qui constitue le travail de thèse d'Aya Jendoubi et qui va être développé dans le reste de cette section.

Dans un SoC, les accès Direct Memory Access (DMA) peuvent être des sources d'attaques étant donné qu'ils sont utilisés par des périphériques pour accéder directement à la mémoire sans utiliser le processeur principal et ses éventuels mécanismes de sécurité. Si des périphériques malveillants utilisent du DMA, ils sont alors en mesure d'accéder à des zones mémoire contenant des données critiques. Pour contrecarrer ces risques, l'IOMMU est un mécanisme qui fournit une isolation des espaces mémoire associés aux périphériques d'entrée-sortie et protège ainsi le système des accès mémoire non autorisés.

Malgré le niveau de protection apporté par l'IOMMU, la flexibilité apportée par des systèmes reconfigurables dynamiquement apporte aussi des nouvelles problématiques de sécurité. La reconfiguration dynamique partielle ou Dynamic Partial Reconfiguration (DPR) permet de modifier les composants matériels pendant l'exécution : néanmoins cette capacité augmente la surface d'attaque en permettant à des entités malveillantes d'exploiter les processus de reconfiguration. Dans le travail de thèse d'Aya Jendoubi, on s'intéresse à une passerelle avec IOMMU qui est en mesure de déployer des accélérateurs matériels selon les besoins de l'utilisateur par l'utilisation de la DPR.

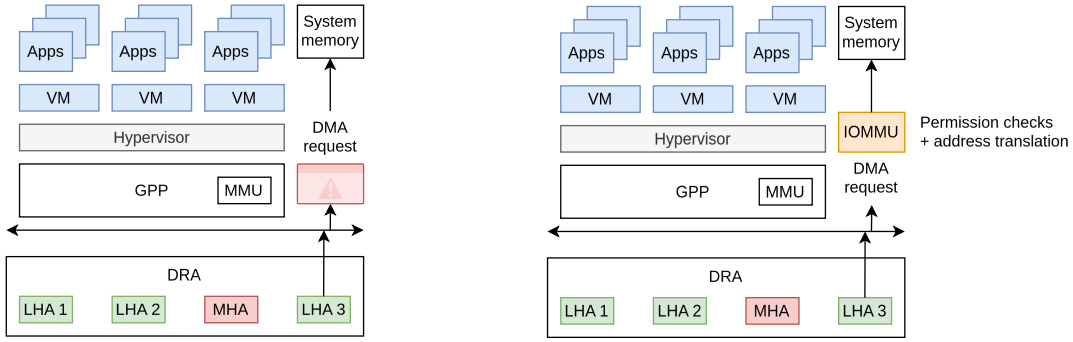
3.1.2 Sécurité des accès mémoire avec IOMMU dans un environnement RISC-V

Dans un objectif de performance, le partage de données entre plusieurs ressources matérielles nécessite souvent du DMA pour optimiser les communications. Cependant, cette configuration permet également à des acteurs malveillants d'exploiter ces accès.

Comme on peut le voir dans la Figure 3.2, l'IOMMU amène des mécanismes de contrôle d'accès qui s'assurent que les périphériques peuvent uniquement accéder aux régions mémoire qui leur sont explicitement allouées : dans cette figure, les MHAs sont des accélérateurs matériels malveillants et les LHAs des accélérateurs matériels légitimes. En isolant les accès mémoires sur différents périphériques, l'IOMMU aide à se prémunir des accès non autorisés et des éventuelles fuites de données.

Cette protection est particulièrement importante dans les systèmes virtualisés où plusieurs VMs partagent les mêmes ressources matérielles. L'IOMMU

1. https://team.inria.fr/sushi/



(a) Sans IOMMU (b) Avec IOMMU

FIGURE 3.2 – Architecture du SoC utilisé dans cette étude : sans et avec IOMMU.

associe les adresses des périphériques à des adresses physiques en mémoire de manière à maintenir une isolation entre les différents environnements virtuels et à atténuer les attaques inter-VM.

L'IOMMU effectue une transformation d'adresses similaire à ce que l'on peut trouver dans une Memory Management Unit (MMU), où on traduit l'Input Output Virtual Address (IOVA) reconnue par le périphérique d'entrée-sortie vers l'adresse physique correspondante. Dans les environnements virtualisés, l'hyperviseur ajoute une couche de traduction, ce qui oblige l'IOMMU à gérer la traduction dite "G-stage" au niveau de l'hyperviseur et la traduction dite "VS-stage" pour chaque VM au niveau du système d'exploitation invité.

3.1.3 État de l'art

3.1.3.1 Analyses de sécurité

Le temps nécessaire à la mise à jour des caches de l'IOMMU augmente le risque d'exploitation par des périphériques malveillants comme cela est évoqué par Duccoussou et al. [43].

De plus, Tiemann et al. [44] ont exposé des vulnérabilités liées aux caches de l'IOMMU, plus particulièrement dans l'Input Output Translation Lookaside Buffer (IOTLB), et apporte une analyse complète de la menace. Les auteurs démontrent comment ces vulnérabilités peuvent faire fuiter des communications et des données sensibles qui transitent entre les périphériques, tels que les FPGAs et les Graphics Processing Units (GPUs), grâce à des fuites temporelles de l'IOTLB qui menacent les environnements *cloud*.

D'autres travaux se sont également intéressé au cas où l'IOMMU utilise la mémoire système pour stocker les tables de translation, cette mémoire pouvant

alors être exploitée par des attaquants pour exfiltrer des données. Par exemple, [45] présentent les défis liés à une acquisition de données fiable dans des environnements *cloud* pour la détection de *malwares* et les analyses forensiques, en particulier dans les systèmes qui utilisent une IOMMU.

D'une manière similaire, Peglow s'est intéressé à la sécurité des plateformes hétérogènes CPU/FPGA utilisant l'IOMMU [46]. Cet article identifie des vulnérabilités spatiales, telle qu'une surexposition de la mémoire liée à des pages mémoire à trop gros grain et des vulnérabilités temporelles dues à des entrées obsolètes dans l'IOTLB. Ce travail s'est tout d'abord intéressé aux systèmes statiques, mais il pose malgré tout les bases sur des problématiques de gestion de cache et de translation d'adresses qui peuvent être pertinentes dans des environnements statiques et dynamiquement reconfigurables.

Bien que n'abordant pas spécifiquement les vulnérabilités liées aux entrées-sorties dans les FPGA multitenants, des travaux tels que [47] et [48] explorent les défis de sécurité qui sont présents dans des environnements avec des FPGAs partagés et qui s'alignent sur notre modèle de menace. Ils proposent des mécanismes pour un contrôle d'accès sécurisé et une isolation par domaines, ce qui offre des solutions pour la gestion des données partagées et la détection des accès non autorisés.

Par ailleurs, [49] discute des vulnérabilités relatives aux ressources partagées et à la reconfiguration dynamique dans des environnements FPGA multitenants, telles que des redirections d'adresses mémoire ou des attaques par dissimulation de tâches. Les auteurs décrivent une solution basée sur un Secure Authentication Module (SAM) qui effectue une vérification en temps réel des tâches matérielles et des applications logicielles par des Multiply and ACcumulates (MACs) qui assurent l'intégrité des tâches et empêchent les modifications de bitstream non autorisées. Cependant, cette solution nécessite d'avoir une configuration statique des tâches et s'appuie en grande partie sur un SAM externe.

Des études telles que celle menée par Morgan et al. [50] expliquent comment intégrer des FPGAs avec des fonctions PCI Express spécifiques et exploiter des vulnérabilités du DMA peut mener à des attaques critiques de mauvaises configurations.

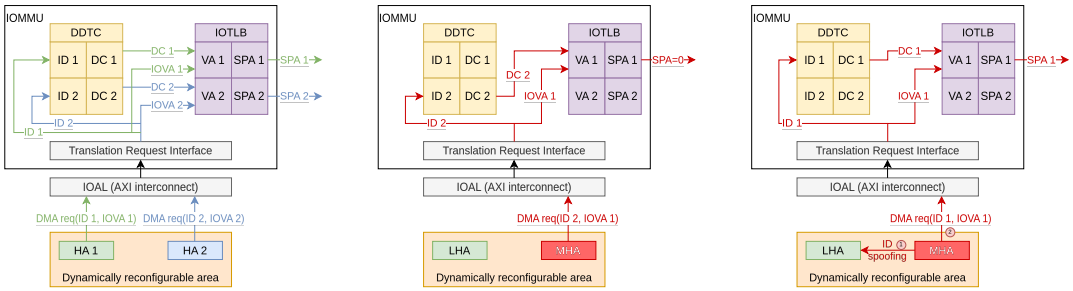
Bossuet et al. [51] démontrent comment des Intellectual Propertys (IPs) malveillantes peuvent effectuer des attaques intra-SoC, y compris des attaques sur la synchronisation des caches et des observations par canaux cachés, pour en déduire des informations sensibles sans équipement externe. Ce travail met aussi en avant des techniques émergentes qui exploitent les signaux du bus Advanced eXtensible Interface (AXI) pour détecter les états du cache et avoir des mécanismes de protection robustes. Ensuite, [52] a présenté plusieurs stratégies d'attaques au niveau Register-Transfer Level (RTL) où les attaquants peuvent manipuler les outils Xilinx pour manipuler les protocoles de sécurité des modules d'inter-

connexion AXI. Cela comprend le fait d'exploiter des IPs d'un monde sécurisé depuis un domain non sécurisé, d'empêcher les accès aux IPs du monde sécurisé, de contourner les erreurs de sécurité, et d'implémenter des attaques basées sur les FIFO pour exfiltrer des données sensibles. Tout ceci met en avant les nombreux problèmes de sécurité présents dans les SoCs hétérogènes.

3.1.3.2 Exploitation de vulnérabilités

Dans notre système basé sur le protocole AXI, la logique qui attribue les accès aux périphériques d'entrée-sortie contrôle les requêtes de lecture et d'écriture depuis les périphériques vers l'IOMMU. Cette fonction d'arbitrage est responsable des requêtes de traduction d'adresse provenant des accélérateurs à travers le Translation Request Interface (TRI) de l'IOMMU qui détermine le séquencement et la priorité à affecter à chaque accès.

Une attaque possible vise particulièrement la nature dynamique de ce processus en usurpant l'identité de périphériques matériels légitimes, contournant ainsi les vérifications effectuées par l'IOMMU.



(a) Les requêtes DMA de deux HAs distincts.

(b) MHA qui échoue à accéder à la région mémoire du LHA.

(c) MHA qui usurpe l'identité du LHA en volant son identifiant et en accédant à sa région mémoire.

FIGURE 3.3 – Scénario d'exploitation : Usurpation de l'identifiant du périphérique (Device ID Spoofing) et accès mémoire non autorisé.

La Figure 3.3 montre qu'en manipulant les entrées du Device Directory Table Cache (DDTC), un acteur malveillant peut obtenir des accès à des zones mémoire contenant des données critiques, compromettant l'isolation proposée par l'IOMMU RISC-V. On expose également des vulnérabilités critiques présentes dans les implémentations actuelles de l'IOMMU, particulièrement dans les systèmes avec une grande flexibilité et des reconfigurations fréquentes.

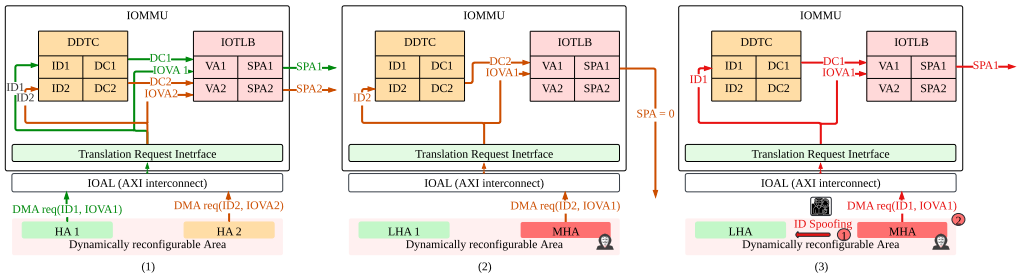


FIGURE 3.4 – Scénario d'exploitation : Usurpation de l'identifiant du périphérique (Device ID Spoofing) et accès mémoire non autorisé (1) Les requêtes DMA de deux HAs distincts (2) MHA qui échoue à accéder à la région mémoire du LHA (3) MHA qui usurpe l'identité du LHA en volant son identifiant et en accédant à sa région mémoire.

3.1.4 Plateforme expérimentale

3.1.4.1 Environnement logiciel et matériel

Pour faire la démonstration d'attaques pouvant être menées par un MHA (voir Figure 3.4), nous avons proposé l'architecture décrite dans la Figure 3.5.

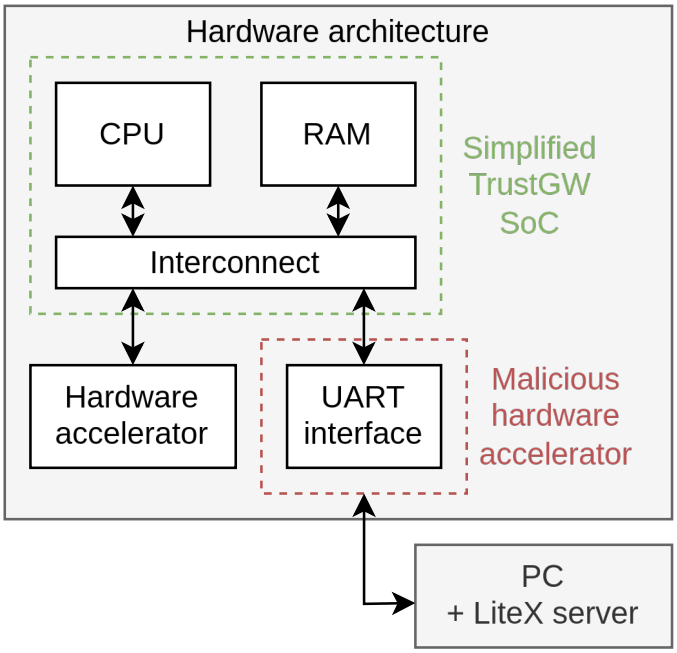


FIGURE 3.5 – Intégration d'un MHA dans un SoC TrustGW.

Un pont Wishbone vers AXI est nécessaire afin de s'interfacer au SoC du projet TrustGW. Pour cette phase de démonstration, ce SoC est réduit à un processeur

CVA6 de l'OpenHW Group sur lequel on exécute l'hyperviseur Bao développé par l'Université de Minho au Portugal [53], [54]. L'hyperviseur contient deux machines virtuelles qui exécutent Linux et FreeRTOS au-dessus de Linux.

Le SoC a été construit avec le framework LiteX [2]. L'accélérateur malveillant est constitué d'un pont entre un périphérique série et un bus Wishbone sur lequel on vient se connecter avec un ordinateur grâce à un outil du framework LiteX (`litex_server`) : ainsi, les attaques sont exécutées depuis la machine de test. Nous avons envisagé 4 scénarios d'attaques dans le cadre de ces travaux de recherche, ils sont brièvement décrits dans la Section 3.1.4.2.

3.1.4.2 Scénarios d'attaques

Démonstration n° 1 L'objectif de la démonstration n° 1 est de montrer qu'un accélérateur malveillant peut lire et écrire dans la mémoire Random Access Memory (RAM) lorsque aucun mécanisme de sécurité n'est mis en place (par exemple l'IOMMU). Le scénario est le suivant :

— Un accélérateur victime écrit et lit dans la mémoire RAM. L'objectif est de simuler des accès DMA.
— Le MHA lit la zone mémoire de l'accélérateur victime.
— Le MHA écrit dans la zone mémoire de l'accélérateur victime.

Démonstration n° 2 Dans la démonstration n° 2, on souhaite montrer qu'un accélérateur malveillant ne peut pas lire et écrire dans la zone mémoire RAM d'un autre accélérateur lorsqu'une IOMMU est présente. Le scénario est le suivant :

— Un accélérateur victime écrit et lit dans la mémoire RAM. L'objectif est de simuler des accès DMA.
— Le MHA essaie de lire dans la zone mémoire de l'accélérateur victime sans succès.
— Le MHA essaie d'écrire dans la zone mémoire de l'accélérateur victime sans succès.

Dans la zone statique, les accélérateurs ont des identifiants fixes ce qui empêche l'usurpation d'identifiants.

Démonstration n° 3 L'objectif de la démonstration n° 3 est de démontrer qu'un accélérateur malveillant déployé dans une zone reconfigurable dynamiquement peut lire et écrire dans la mémoire RAM d'un autre accélérateur alors qu'une IOMMU est présente. Nous avons également comme hypothèse que l'attaquant a réussi à récupérer l'identifiant de sa victime.

2. https://github.com/enjoy-digital/litex

Le scénario est le suivant :
— Un accélérateur victime écrit et lit dans la mémoire RAM. L'objectif est de simuler des accès DMA.
— Le MHA peut lire dans la zone mémoire de l'accélérateur victime avec succès.
— Le MHA peut écrire dans la zone mémoire de l'accélérateur victime avec succès.

Démonstration n° 4 L'objectif de la démonstration n° 4 est de démontrer qu'un accélérateur malveillant déployé dans une zone reconfigurable dynamiquement ne peut pas lire ni écrire dans la mémoire RAM d'un autre accélérateur alors qu'un durcissement fort du système est utilisé : l'IOMMU et le wrapper permettant de faire du contrôle d'accès. Comme dans la démonstration n° 3, on considère également que l'attaquant a réussi à récupérer l'identifiant de sa victime. Le scénario est le suivant :
— Un accélérateur victime écrit et lit dans la mémoire RAM. L'objectif est de simuler des accès DMA.
— Le MHA ne peut pas lire dans la zone mémoire de l'accélérateur victime.
— Le MHA ne peut écrire pas lire dans la zone mémoire de l'accélérateur victime.

3.2 Définition d'architectures d'IA pour de la détection d'intrusion

Cette section décrit des travaux en cours pour la thèse de Pierre Garreau qui a débutée fin 2023 dans le cadre de la chaire de cyberdéfense des systèmes navals (https://chaire-cyber-navale.fr/). Dans cette thèse, la sécurité est abordée à un niveau d'abstraction plus élevé. En effet, nous souhaitons étudier comment réaliser un système de détection d'intrusions (ou Intrusion Detection System (IDS)) pour détecter des intrusions dans une meute d'appareils malveillants à l'aide d'algorithmes d'intelligence artificielle frugaux. L'objectif final est d'implémenter un système sur un composant reconfigurable FPGA avec un processeur RISC-V et des accélérateurs matériels dédiés à l'Intelligence Artificielle (IA). Le reste de la section donne quelques éléments de l'état sur les algorithmes et les accélérations matérielles possibles.

3.2.1 Contexte du projet

De nos jours, les véhicules autonomes sont de plus en plus présents dans les forces militaires. Par exemple, l'entreprise allemande ARX va investir plusieurs

millions d'euros pour construire une nouvelle usine de véhicules terrestres autonomes [55] ou Overland AI, une startup américaine qui développe un véhicule tout-terrain autonome [55]. Dans un contexte plus maritime, on peut également citer Naval Group [56] ou Thales [57] qui proposent des produits autonomes pouvant évoluer sur et sous l'eau.

De plus, avec l'explosion des cyberattaques, il est désormais indispensable de sécuriser les communications entre une flotte de véhicules autonomes et le vaisseau-mère. L'apprentissage automatique (ou Machine Learning (ML) dans la suite de ce manuscrit) propose désormais des modèles qui sont en mesure de réaliser des détections d'intrusions efficaces. Cependant, la majorité de ces modèles nécessite une puissance de calcul et des besoins en mémoire conséquents.

Les systèmes embarqués sur les drones ont des contraintes fortes sur la puissance et la mémoire disponible afin de maximiser leur autonomie : cela empêche d'intégrer des modèles classiques de ML qui sont gourmands. En prenant en compte ces contraintes, nous souhaitons étudier les architectures de Machine Learning qui pourraient être implémentées dans un système comprenant un processeur RISC-V dont le jeu d'instructions est facilement extensible afin de proposer des extensions qui permettraient d'accélérer les calculs liés aux algorithmes d'intelligence artificielle.

3.2.2 L'intelligence artificielle au service de la détection d'intrusions

Depuis quelques années, des travaux de recherche ont étudié l'application de l'intelligence artificielle pour de la détection d'intrusions. Ferrag et al. [58] présente plusieurs techniques d'apprentissage profond pour la détection d'intrusions. Les auteurs séparent les 7 modèles étudiés en 2 catégories distinctes : d'une part, les modèles discriminants ; d'autre part, les modèles génératifs ou non supervisés.

La première catégorie comprend les Deep Neural Networks (DNNs), les Recurrent Neural Networks (RNNs) et les Convolutional Neural Networks (CNNs). Ces modèles sont souvent utilisés dans le Machine Learning et donne des résultats décents pour des applications d'IDS comme le montre l'article de Ferrag et al. [58]. La deuxième catégorie inclut les Restricted Boltzmann Machines (RBMs), les Deep Belief Networks (DBNs), les Deep Boltzmann Machines (DBMs) et les Deep Autoencoders (DAs).

D'autres techniques de Machine Learning ont également été explorées. Par exemple, Bouazzati et al. [59] implémentent un IDS basé sur du Reinforcement Learning (RL) en utilisant de l'apprentissage hors-ligne étant donné que l'apprentissage en ligne augmenterait significativement la consommation du système. Farnaaz et Jabbar [60] présentent une solution basée sur un modèle de

type Random Forest (RF) : selon les auteurs, cette technique a des résultats qui dépassent les IDS traditionnels n'étant pas basés sur de l'intelligence artificielle : En effet, comme le précise Choudhary et al. [61], ces derniers sont principalement basés sur de la reconnaissance de signatures et sont régulièrement dépassés par les techniques qui détectent des anomalies.

De plus, le fait d'intégrer de la détection d'intrusions dans un environnement avec une meute de drones nous incite à aller étudier également les IDS avec de l'apprentissage fédéré (ou Federated Learning (FL)) ou multiagents [62].

3.2.3 Accélération de fonctions de Machine Learning

L'accélération des fonctions de Machine Learning peut se faire de 4 façons différentes :

- L'accélération de fonctions spécialisées.
- Des extensions de jeux d'instructions.
- Par l'optimisation des transferts de données.
- Par des approches de coconception logicielle-matérielle.

À l'heure où j'écris ce manuscrit, un article de type *survey* est en cours d'évaluation dans la revue Elsevier "Microprocessors and Microsystems". Cet article décrit en détail ces 4 approches et les contributions principales pour chacune. Je propose dans la suite de cette sous-section de s'intéresser plus particulièrement à 2 d'entre elles que nous devrions explorer davantage dans la suite des travaux de la thèse de Pierre Garreau.

3.2.3.1 Extensions de jeux d'instructions

Les architectures basées sur le jeu d'instructions RISC-V peuvent se révéler intéressantes dans ce cadre. Étant donné qu'il est extensible, il est possible d'implémenter des modules matériels spécialisés dans les calculs gourmands qu'on retrouve dans les modèles de Machine Learning. Les CNNs sont les modèles les plus couramment accélérés par du matériel. L'opération de convolution est particulièrement gourmande. Pour répondre à cette problématique, Kovacevic et al. [63] ont développé un processeur RISC-V qui inclut un processeur scalaire et un processeur vectoriel. Les deux coeurs fonctionnent simultanément et optimisent la charge de travail de sorte que les calculs vectoriels et matriciels, tels que les opérations de convolution, soient exécutés par le coeur spécialisé en vectoriel. Wu et al. [64] ont ajouté des instructions RISC-V pour lier un coprocesseur au processeur principal et l'utiliser pour les calculs relatifs à la convolution.

Les DNNs peuvent également être accélérés matériellement. Par exemple, Askarihemmat et al. [65] ont implémenté un accélérateur nommé BARVINN [3] qui comprend des Matrix Vector Units (MVUs) qui sont des modules matériels spécialisés dans les calculs de type GEneral Matrix-Vector multiplication (GEMV), GEneral Matrix Multiplication (GEMM) ou encore de type convolution. Vermat et al. ont également développé un accélérateur en utilisant de nouvelles instructions, mais proposent en plus une méthode complète pour optimiser la consommation de leur solution.

L'accélération d'algorithme de Machine Learning peut se faire par la quantification. Cette technique a pour but de réduire le temps de calcul en optimisant la précision des données utilisées dans le traitement. Sanchez-Flores et al. [66] ont modifié les poids de leur CNN avec de la précision multiple pour optimiser l'utilisation de la mémoire et son temps d'exécution. Dès lors qu'on arrive à un compromis entre la précision du modèle et les ressources utilisées, des modèles plus larges peuvent être intégrés dans des systèmes embarqués.

3.2.3.2 Optimisation des transferts de données

Au-delà de l'accélération des calculs, l'optimisation des transferts de données a également été identifiée comme critique pour optimiser les temps d'exécution des algorithmes de Machine Learning. La solution la plus couramment employée est d'ajouter des buffers dans les modules de calculs. Plus récemment, le Processing-in-Memory (PiM) est apparu comme une alternative intéressante. L'objectif général des solutions de PiM est de mettre l'unité de calcul au plus près des éléments de mémoire : ainsi, on évite le transfert dans des registres intermédiaires côté processeur qui ajouteraient de la latence non désirée.

Par exemple, Bavikadi et al. [67] proposent une architecture PiM avec un cluster de calcul dont les dimensions sont reconfigurables dont les éléments principaux qui réalisent des opérations spécifiques aux CNNs sont placés au plus près de la mémoire. Dans une autre approche, Zhang et al. [68] présentent un accélérateur pour des opérations d'inférences dans les DNNs où les éléments de calcul sont directement intégrés dans le processeur. Ces deux exemples de travaux sont adaptés à l'utilisation d'un modèle unique. Lorsque plusieurs modèles doivent être traités en parallèle, d'autres solutions existent pour optimiser l'ordonnancement des calculs : par exemple, Choi et al. [69], [70] ont proposé deux solutions qui permettent de gérer plusieurs DNNs en parallèle.

3. https://github.com/hossein1387/BARVINN

3.3 Conclusion et perspectives

Ce chapitre est une synthèse des travaux de la thèse d'Aya Jendoubi sur le projet ANR TrustGW et de celle de Pierre Garreau financée par la chaire de cyberdéfense des systèmes navals. Les publications associées à ces deux thèses sont présentées en annexe A.2.

Hyperviseurs et microarchitecture

À l'heure de l'écriture de ce manuscrit, les scénarios sont en cours de développement. Dans le cadre du projet ANR TrustGW, il y a un lot de travail dont l'objectif est de livrer un démonstrateur global avec les contributions des différentes thèses, le travail d'Aya Jendoubi y sera bien entendu intégré. Au cours de cette thèse et des autres travaux de recherche que j'ai menés en parallèle, j'ai pu identifier des pistes qui me paraissent intéressantes à étudier à l'avenir.

Le Chapitre 4 présente quelques travaux passés et en cours sur des problématiques de sécurité au niveau micro-architectural. J'imagine naturellement une étude autour de la sécurité de Bao à cette échelle afin de répondre à des questions autour des canaux cachés créés par l'hyperviseur et le contournement de l'isolation des VMs en utilisant des métriques issues du processeur (par exemple, avec des compteurs de performance). D'autre part, on voit que l'implémentation d'une IOMMU dépend de l'utilisation de plusieurs identifiants et a un impact sur la façon dont on utilise les signaux du protocole AXI. Zonta et al. [71] ont publié une étude qui démontre que les implémentations du protocole AXI ne suivent pas toutes les caractéristiques fonctionnelles présentes dans les spécifications du protocole. À partir de ce constat, il est possible d'exploiter certaines des faiblesses proposées dans l'article de Zonta et al. [72] pour exposer des attaques sur certaines implémentations de blocs d'interconnexion AXI, notamment au niveau de la microarchitecture. L'interaction entre la couche logicielle et les vulnérabilités du bus de données pourraient alors faire l'objet d'études ultérieures.

Accélérateurs d'IA sécurisés

Les travaux initiés dans la thèse de Pierre Garreau présentés dans la Section 3.2 sont toujours en cours. Nous en sommes à un point où l'état de l'art est suffisamment abouti pour envisager l'implémentation d'une solution qui permettra à terme de faire de la détection d'intrusion reconfigurable et efficace. Les contours exacts sont encore à préciser, mais nous devrions nous diriger vers une solution avec un processeur RISC-V existant et un coprocesseur implémentant différents accélérateurs matériels pour pouvoir utiliser différents modèles de Machine Learning à la demande. Nous envisageons une preuve de concept qui serait sous la

forme suivante :

- Une architecture avec deux modèles : un pour de la reconnaissance d'image, l'autre pour de l'IDS.
- Chaque modèle aurait deux variantes :
  - Une à efficacité maximale et consommation importante.
  - Une autre orientée basse consommation, mais avec une efficacité altérée.

L'architecture envisagée serait basée sur un SoC tel que Pulpissimo [73] qui utilise un coeur principal RISC-V (RI5CY ou Ibex) et une IP nommée Hardware Processing Engine (HWPE) qui permet d'implémenter des accélérateurs matériels [74]. Dans notre cas, les accélérateurs seraient les modèles décrits précédemment. Ensuite, nous aimerions récupérer des métriques sur la consommation d'une telle architecture afin de proposer un composant qui pourrait facilement permuter entre les variantes de chaque modèle afin de répondre aux contraintes globales du circuit.

Chapitre 4

Contributions sur la sécurité au niveau micro-architecture

Sommaire

Dans les chapitres précédents, j'ai présenté des contributions proches du matériel (Chapitre 3) et d'autres contributions basées sur une conception conjointe logicielle-matérielle (Chapitre 2). Dans ce nouveau chapitre, je souhaite également évoqué la sécurité au niveau micro-architectural : en effet, lorsqu'on déploie un système sur puce, le système lui-même peut être source de fuites pouvant mener à des failles de sécurité. Ce chapitre évoquera notamment des contextes où les mémoires cache sont des cibles privilégiées.

4.1 Protection des mémoires caches contre les *timing attacks*

4.1.1 Micro-architecture d'un système embarqué

La Figure 4.1 présente l'architecture interne du processeur CV32E40P auquel un niveau de cache ont été ajoutés.

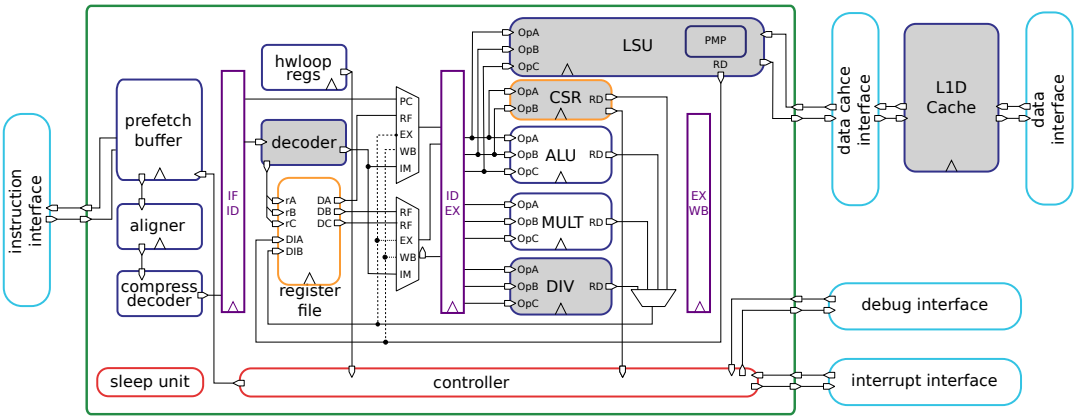


FIGURE 4.1 – Architecture d'un SoC basée sur le processeur CV32E40P avec un niveau de cache.

Son pipeline est composé de 4 étages qu'on retrouve dans la majorité des processeurs : *Fetch (IF)* pour lire les instructions, *Decode (ID)* pour les décoder, *EXecute (EX)* pour exécuter les instructions et enfin *WriteBack (WB)* pour écrire le résultat de l'instruction dans le banc de registres internes au processeur.

La mémoire cache présente dans la partie droite de la Figure 4.1 sert à diminuer la latence lors des accès mémoires : la mémoire cache étant placée entre le Central Processing Unit (CPU) et la mémoire externe, si la donnée est présente en cache, le processeur n'ira pas la chercher plus loin.

Les caches peuvent se classer en plusieurs catégories :

- Caches à correspondance (*direct-mapped cache*) : chaque ligne de cache est associée à une adresse en fonction d'une partie de cette dernière qu'on appelle index.
- Caches associatifs (*fully associative cache*) : une adresse peut se localiser dans n'importe quelle ligne de cache. La correspondance adresse - ligne de cache est réalisée par une politique de remplacement.
- Caches associatifs à N voies (*N-way set associative cache*) : la structure la plus répandue (voir Figure 4.2) sous forme d'une matrice de S ensembles (ou *sets*) de N voies (ou *ways*). Chaque adresse est associée à un set en fonction de son index.

Dans la Figure 4.2, une adresse de A bits est décomposée en trois parties : un tag, un index et un offset. Le tag est l'offset permettent d'identifier les octets à manipuler dans la ligne de cache et de différencier les différentes adresses dans un même cache set. Dans le Chapitre 4, la politique Least Recently Used (LRU) est utilisée : celle-ci évince la voie la moins récemment utilisée dans le cache set.

Nous nous sommes intéressés aux attaques par canaux auxiliaires. Ces attaques sont basées sur l'observation de sources d'information extraites pendant

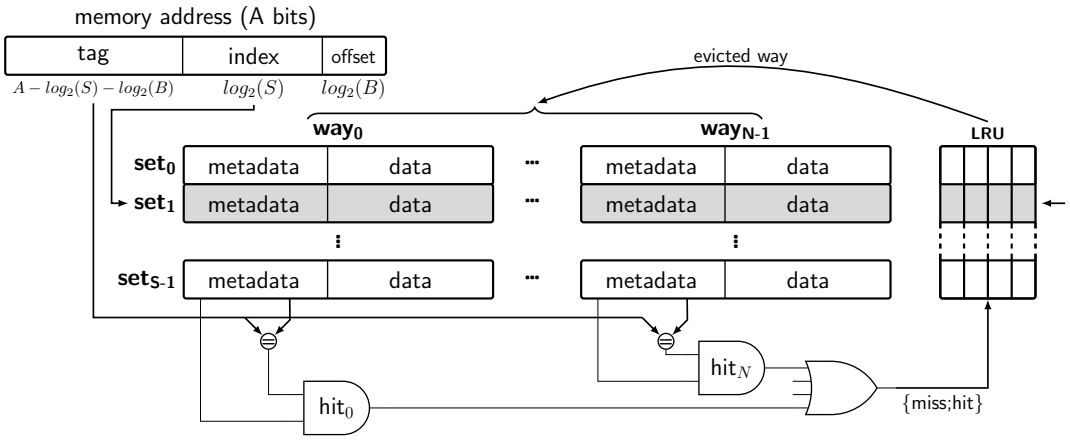


FIGURE 4.2 – Cache associatif à N voies.

l'exécution du programme pour en extraire des informations critiques telles que des clés cryptographiques ou des mots de passe. Dans le Chapitre 4, on se concentre plus particulièrement sur les fuites temporelles : mesure du temps d'exécution de la victime, étude des mécanismes d'optimisation et de partage de ressources, etc. Lorsqu'on s'intéresse au cache, on se base sur le fait que le temps d'accès à une donnée est fonction de sa position dans la hiérarchie mémoire.

Les articles [75], [76] donnent une liste exhaustive d'attaques sur les caches. Par exemple, FLUSH+RELOAD [77] utilise une instruction du jeu d'instruction x86 pour évincer des données du cache vers le LLC. Plus tard, l'attaquant peut accéder aux ressources partagées dont l'adresse dépend du secret pour mesurer le temps d'accès et en déduire les données auxquelles la victime a accédé.

On peut citer également PRIME+PROBE [78]-[81] est une attaque qui possède plusieurs variantes afin de s'attaquer aux différents niveaux de cache. Cette attaque se déroule en trois étapes : 1) l'attaquant remplit son ensemble d'éviction avec des données qui lui sont propres ; 2) dans un second temps, la victime exécute son code et va écraser certaines données (ce qui impose que l'ensemble d'éviction choisi par l'attaquant soit bien en phase avec les données de la victime) ; 3) l'attaquant pourra alors mesurer le temps d'accès aux différentes données : si le temps d'accès est court, la donnée n'a pas été évincée et si ce temps est plus long, la donnée a été évincée par une donnée de la victime.

PRIME+PRUNE+PROBE [82] est une évolution de PRIME+PROBE qui nécessite moins d'accès pour construire l'ensemble d'éviction. De plus, les contremesures basées sur l'ajout d'aléa sont moins efficaces contre cette catégorie d'attaques.

Enfin, EVICT+TIME [80] est une autre attaque qui mesure le temps d'exécution de la victime et qui se déroule en trois étapes : 1) l'attaquant mesure le temps d'exécution de la victime ; 2) l'attaquant réalise l'étape PRIME d'un

PRIME+PROBE pour évincer les données de la victime ; 3) enfin, l'attaquant mesure de nouveau le temps d'exécution de la victime : si le temps diffère, la victime a accédé au cache set manipulé pendant la phase de PRIME.

Plusieurs contremesures ont été proposées contre ces attaques sur les fuites temporelles : celles-ci peuvent être implémentées sous forme de modifications de la couche logicielle, de modifications du matériel sur lequel s'exécute le programme, voire d'une proposition hybride logicielle/matérielle.

Ces contre-mesures peuvent se classer en trois grandes catégories.

Tout d'abord, on peut réaliser une détection qui peut se matérialiser de différentes manières. Elle peut se faire directement en analysant le programme pour le rendre indépendant du temps. Autrement dit, on cherche à faire de la programmation dite "temps constant" (*constant time programming*) : c'est par exemple le cas de Wu et al. [83] qui proposent une analyse des accès mémoire pour améliorer cette exécution en temps constant.

D'autre part, certaines techniques ajoutent de l'aléa (*randomization*) afin de perturber l'attaquant qui aura plus de difficultés pour construire son ensemble d'éviction. Des travaux tels que RPCache [84] ou CEASER [85] proposent des solutions qui permettent de manipuler l'index qui assigne un ensemble de ligne de cache à une adresse donnée. D'autres travaux tels que CEASER-S [86] et ScatterCache [87] modifient quant à eux la façon dont on peut indexer une voie ou un groupe de voies.

Ensuite, d'autres contremesures proposent de faire un partitionnement soit temporel, soit matériel. Pour le partitionnement temporel, on identifie par exemple, des solutions qui étendent le jeu d'instruction pour isoler temporellement les applications [88], [89]. D'autre part, les travaux avec des impacts sur le matériel vont s'intéresser à un partitionnement par voie (NoMocache [90]) voire par ligne de cache (PLcache [84]) qui est la granularité la plus fine qu'on puisse manipuler dans ce contexte.

4.1.2 Contexte du projet SCRATCHS

Les travaux évoqués dans la Section 4.1 font écho aux travaux de la thèse de Nicolas Gaudin qui s'est déroulée dans le cadre du projet Side-Channel Resistant Applications Through Co-designed Hardware/Software (SCRATCHS). Dans ce projet, nous souhaitions mettre en place des protections contre les attaques par canaux cachés temporels qui visent les mémoires cache pour des systèmes embarqués à base d'architecture RISC-V. Ce projet a été le cadre d'une deuxième thèse qui s'intéressait plus particulièrement à des solutions principalement logicielles [91].

La Figure 4.3 présente le système envisagé dans ce projet et dans la suite de cette section : deux applications qui s'exécutent sur un processeur monocoeur,

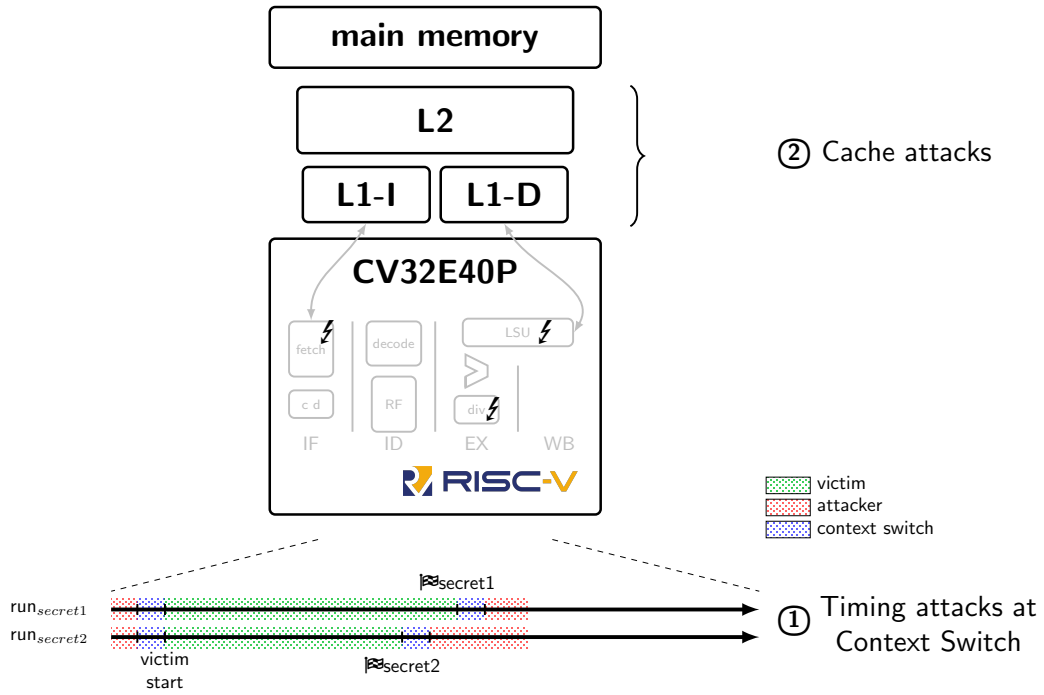


FIGURE 4.3 – Modèle de menaces.

une hiérarchie de cache simple, la pile logicielle est de confiance et l'attaquant peut inférer l'état du cache avant et après l'exécution de la victime tout en étant capable de mesurer les temps d'exécution avec les compteurs de cycles disponibles dans le processeur (ici, le processeur CV32E40P de l'OpenHW Group).

La suite de la Section 4.1 décrit la solution avec le mécanisme de verrouillage de données étudiée dans la thèse de Nicolas Gaudin.

4.1.3 `lock/unlock`, un mécanisme de verrouillage de lignes de cache

Dans les travaux existants, PLcache [84] propose déjà de verrouiller des lignes de caches avec des instructions `lock` et `unlock` : une ligne verrouillée ne peut être évincée par aucun process. Cependant, cette solution a ses faiblesses :

- Selon l'état du cache et celui des métadonnées de la politique de remplacement, il est malgré tout possible d'extraire les données présentes dans les lignes de cache verrouillées.
- Un accès mémoire peut contourner le mécanisme quand la politique de remplacement vise une ligne de cache verrouillée par un autre processus.

L'utilisation d'un mécanisme de verrouillage permet d'optimiser l'utilisation du cache en protégeant un sous-ensemble de l'espace mémoire disponible. Cette

approche permet que le process protégé libère les lignes verrouillées le plus vite possible et puisse continuer son exécution. De plus, pour éviter le problème de contournement, on conserve une ligne de cache (par conséquent non verrouillée) afin de conserver des performances acceptables pour les applications en cours d'exécution.

Si on considère un cache associatif à N voies et la LRU originale, la métadonnée associée à une voie peut être associée à N états (autrement dit, pour un cache associatif à 4 voies, les états vont de 1 à 4). L'état est une représentation de l'âge de la ligne de cache : 1 pour la ligne utilisée la plus récemment, 4 pour la plus ancienne. La solution présentée dans cette section ajoute un état pour les lignes de cache verrouillées (état égal à 0) : pour un cache à N voies, il faudra donc $N + 1$ états.

La Figure 4.4 présente un cas d'étude simple pour illustrer le comportement de la LRU pour trois accès mémoire consécutifs : un verrouillage (`lock`), un accès classique (`lw` et un déverrouillage (`unlock`)) où les adresses sont dans le même ensemble.

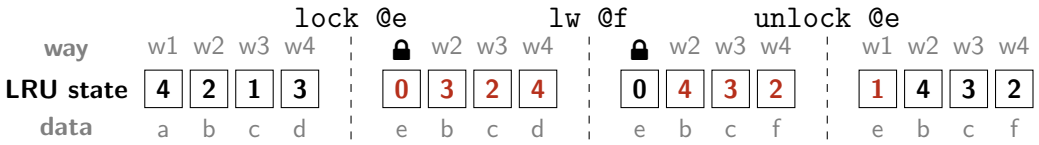


FIGURE 4.4 – Cas d'étude.

Cette figure présente les métadonnées de la politique de remplacement (états LRU ou *LRU states*) pour chaque voie d'un ensemble de lignes de cache. L'ensemble est composé de $N = 4$ voies `wX` qui sont déjà utilisées. Avant de prendre en compte l'accès `lock @e`., nous pouvons noter que la voie la moins récemment utilisée est `w1` (son état LRU est égal à 4). De plus, il n'y a pas de ligne verrouillée ($N_{lock} = 0$ et états LRU $> N_{lock} + 1$ pour toutes les voies).

Ensuite, après avoir exécuté l'instruction `lock @e`, la donnée `a` a été évincée et la voie `w1` est désormais verrouillée (état égal à 0). De plus, les états des autres voies non verrouillées ont été mis à jour : il reste trois candidats LRU (état $> N_{lock} + 1$ et $N_{lock} = 1$). Comme $w1$ est verrouillée, elle ne peut pas être sélectionnée par la politique de remplacement.

Après avoir chargé un mot à l'adresse `f` (`lw @f`), la donnée `d` a été évincée par la donnée `f` et la voie `w4` devient la plus récemment utilisée (état égal à 2). Pendant ce temps, les états des voies `w2` et `w3` ont été mis à jour alors que la voie `w1` est restée verrouillée. Il y a toujours trois candidats pour la politique de remplacement. Finalement, `w1` est déverrouillée en exécutant l'instruction `unlock @e` et a été remise comme une candidate potentielle en étant la plus récemment

utilisée. Du côté du logiciel, la personne qui souhaite utiliser ce mécanisme doit l'appeler explicitement dans son code, comme cela est présenté dans le listing 4.1 et 4.2 pour le code assembleur de la fonction de verrouillage.

```
void fct(int* sensitive_table, int* input){
  // Phase de verrouillage
  for(int i=0; i<sizeof(sensitive_table); i+=1)
      lock_macro(&sensitive_table, i);

  // La fonction algo accède à la table dépendante du secret
  algo(sensitive_table, input);

  // Phase de déverrouillage
  for(int i=0; i<sizeof(sensitive_table); i+=1)
      unlock_macro(&sensitive_table, i);
}
```

Listing 4.1 – Exemple d'utilisation des instructions `lock` et `unlock`.

```
c.mv  t4,a4 # déplacer l'adresse &table dans t4
c.mv  t5,a5 # déplacer i dans t5
c.add t4,t5
lock  x0,0(t4)
```

Listing 4.2 – Code assembleur de la macro de verrouillage `lock_macro`.

On aurait pu utiliser des techniques d'annotation [92] et de propagation de teintes (comme cela a été évoqué dans la Section 2.1 ou dans l'article [93]) pour faciliter l'insertion et le suivi de ces instructions.

4.1.3.1 Implémentation et résultats

Les développements liés à ce mécanisme de verrouillage de cache ont été implémentés sur un processeur CV32E40P (voir Figure 4.1) qui implémente a minima les extensions I, M et C du jeu d'instructions RISC-V. Les fuites temporelles sur la micro-architecture du processeur elles-mêmes ne sont pas décrites dans ce manuscrit même s'il en existe (par exemple, le temps d'exécution de l'algorithme de division est dépendant de la taille des opérandes).

Les métadonnées des différentes voies du cache sont mises à jour à chaque accès mémoire. Pour prendre en compte les instructions `lock` et `unlock`, il a fallu étendre la micro-architecture du CV32E40P (notamment pour décoder ces instructions), modifier le cache L1D ainsi que proposer une nouvelle chaîne de compilation intégrant ces nouvelles instructions.

Le module implémentant la LRU intégrant le mécanisme de verrouillage est composée de trois modules principaux :

- *LRU state* : calcule la valeur d'état minimal en fonction du nombre de lignes de cache verrouillées et de l'instruction en cours d'exécution. Il déclenche une exception lorsqu'on souhaite verrouiller un cache où toutes les lignes le sont déjà.
- *Update way* et *Update accessed way*. Dans le cache où nous avons des ensembles de 4 voies, le mécanisme propose 3 modules *update way* et 1 module *update accessed way* afin de mettre à jour l'état des différentes voies.

4.1.3.2 Résultats

L'architecture composée du processeur et du cache a été implémentée sur un FPGA Xilinx Kintex-7 et avec les outils Vivado 2022.2. La Table 4.1 présente les résultats en surface du processeur et du cache dans deux versions : *baseline* qui est la version de référence et *protected* qui est la version intégrant le mécanisme de verrouillage de cache.

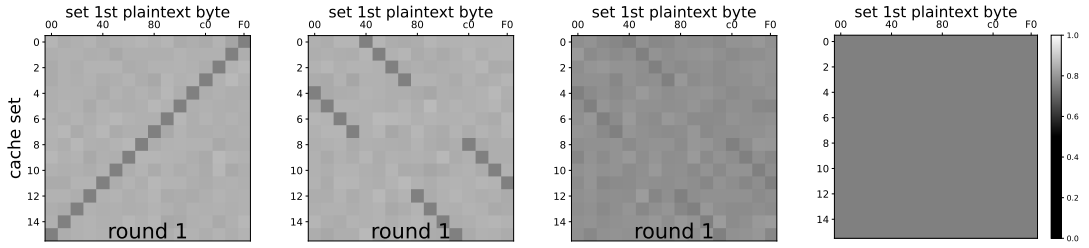
Table 4.1 – Résulats de synthèse pour un FPGA Kintex-7.

		Cache	**CPU**
Baseline	LUTs	980	5 661
	FFs	1 065	3 465
	BRAMs	8,5	8,5
Protected	LUTs	1 007 (+ 2,8%)	5 683 (+ 0,7%)
	FFs	1 077 (+ 1,1%)	3 481 (+ 0,3%)
	BRAMs	8,5	8,5

On remarque qu'on utilise le même nombre de Block RAMs (BRAMs) (blocs mémoire qui composent le cache et servent de stockage aux métadonnées de la LRU). Les surcoûts en surface sont quant à eux négligeables d'autant plus que nous avons utilisé un processeur relativement simple.

Ensuite, nous avons souhaité réaliser une évaluation de la sécurité de notre solution. Dans notre cas d'étude, nous avons pris en compte l'attaque

Prime+Probe sur un algorithme AES-128. Plus précisément, nous nous sommes intéressés au premier octet de la clé de chiffrement. Nous considérons le cas d'une attaque en clair connu et où les accès à la S-Box dépendent du message en clair et de la clé de chiffrement. La Figure 4.5 présente une analyse Prime+Probe dans différents cas : plus le carré est foncé, plus la victime a évincé les données de l'attaquant entre la phase de Prime et la phase de Probe.



(a) Première ronde, clé=`0xFF`. (b) Première ronde, clé=`0x42`. (c) Deuxième ronde, clé=`0x42`. (d) S-Box verrouillée en cache, clé=`0x42`.

Figure 4.5 – Cartographie des résultats d'attaques en caches menées sur AES-128.

Dans les Figures 4.5a et 4.5b, on observe des motifs facilement identifiables après 1 ronde d'AES pour un premier octet de clé fixé à `0xFF` et `0x42`. Toujours avec un premier octet fixé à `0x42`, la Figure 4.5c montre le résultat après 2 rondes d'AES : le motif est toujours visible, mais le contraste entre les carrés les plus foncés et les plus claires est moindre : ceci est dû à la diffusion de l'algorithme de chiffrement. Par contre, lorsque la S-Box est verrouillée en cache comme dans la Figure 4.5d, la couleur est uniforme : en d'autres termes, l'attaquant ne peut déduire aucune information.

Enfin, la Table 4.2 présente les pénalités en performance et les surcoûts en surface de travaux de l'état de l'art s'intéressant aux fuites temporelles avec des attaques en cache.

Table 4.2 – Comparaison avec d'autres solutions existantes.

Contremesure	Mécanisme	Type	Temps constant	Surcoûts	
				Performance	Surface
Winderix et al. [94]	Compiler-Assisted Hardening	Logiciel	Oui	19-76%	0%
[84], [85], [95]	Randomization	Matériel	Non	1-10%	5-10%
NoMocache [90]	Fixed way partitioning	Matériel	Non	5%	-
SecDCP [96]	Dynamic way partitioning	Matériel	Non	12%	-
PLcache [84]	Dynamic cache line partitioning	Hybride	Non	12%	-
Notre contribution	Dynamic cache line partitioning	Hybride	Oui	2%	$< 3\%$

Winderix et al. [94] présentent une solution purement logicielle où le compilateur est modifié pour rendre le code à temps constant, le surcoût temporel y

est important (entre 19 et 76%) mais ne nécessite pas de modification du matériel. Les travaux utilisant de la randomisation [84], [85], [95] ont un surcoût en performance inférieur à 10% et un surcoût en surface de 5 à 10%. Les techniques de partitionnement telles que NoMocache [90] ou SecDCP [96] font en sorte que les voies soient allouées à un processus ou un ensemble de processus : la pénalité en performance y reste convenable (inférieure à 12%). La solution la plus proche de notre travail est PLcache mais celle-ci n'est pas à temps constant et dégrade les performances de façon plus importante.

4.1.4 Une solution "hybride"

Les travaux présentés jusqu'à maintenant dans la Section 4.1 sont un sous-ensemble des mécanismes de sécurité qui cherchent à protéger les caches. D'autres travaux, basés sur une indexation asymétrique des voies comparent les traces avant et après l'éviction de données dans un cache set pour contrer les attaques sur le temps d'exécution. Un aléa est ajouté entre l'adresse et les index accédés dans chaque voie. Le renouvellement de la génération d'aléa est un élément critique pour garantir la sécurité de ce type de solutions, mais implique un surcoût en performances important (il invalide l'ensemble du cache). De plus, ce renouvellement est spécifique à une fonction de dérivation d'index (ou Index Derivation Function (IDF)) utilisée qui est basée sur une clé.

Nous avons donc proposé une solution hybride combinant de la randomisation et du partitionnement pour garantir l'exécution en temps constant et également empêcher la fuite du secret lorsque l'application verrouille un nombre de lignes dépendant du secret. La Figure 4.6 présente le principe de notre solution dans laquelle une gestion asymétrique des voies est couplée au mécanisme de verrouillage de lignes de cache.

Une IDF `f` est assignée à chaque voie du cache. Chaque IDF prend comme entrée tout ou partie de l'adresse ainsi qu'une clé. L'attribution d'une clé unique pour chaque IDF crée l'asymétrie dans le cache. On a choisi l'IDF SCARF [97] car sa latence et son coût en surface sont faibles.

Nicolas Gaudin, le doctorant que j'ai co-encadré dans le projet SCRATCHS, a effectué une mobilité à l'Université de la Ruhr à Bochum. Les travaux effectués en Allemagne ont menés à une publication à la conférence AsiaCCS [98] dans laquelle les auteurs ont analysé différentes politiques de cache dans un contexte de réindexation asymétrique sur une attaque Prime+Prune+Probe. Nous avons également regardé différentes politiques de remplacement, nous sommes partis sur VARP qui manipule l'âge de la ligne de cache Cette nouvelle politique de sécurité implique des modifications sur les mécanismes matériels de mise à jour de voies. Les mécanismes sont décrits plus en détails dans l'article publié à AsiaCCS en 2024 [98].

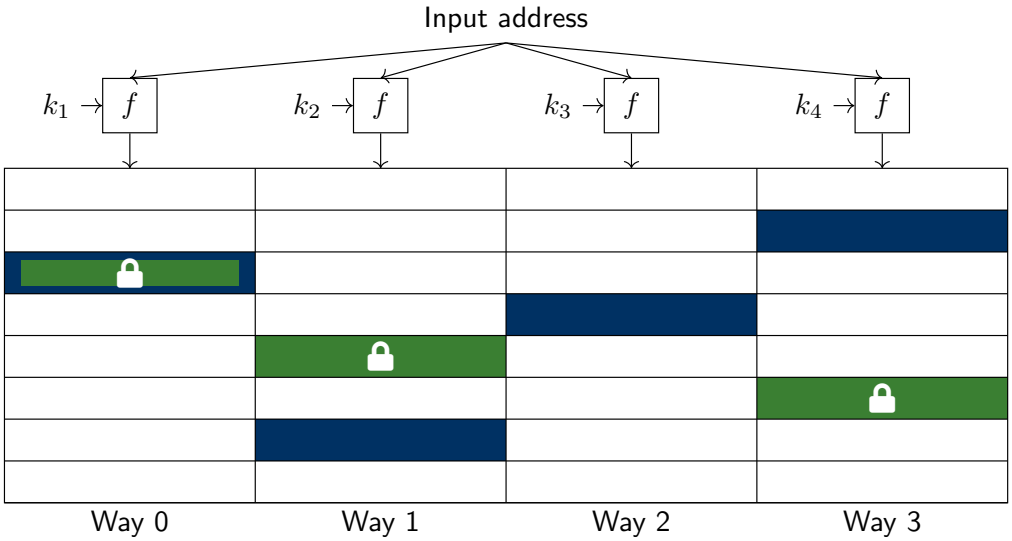


FIGURE 4.6 – Schéma bloc du cache hybride implémentant le mécanisme de verrouillage.

Cette solution hybride a été implémentée sur un composant FPGA Kintex-7 avec les outils AMD Vivado 2022.2. La Table 4.3 présente les résultats de synthèse associés : on y trouve le CPU, le coeur ainsi que le coût en surface de la politique de remplacement pour les caches d'instructions et de données.

TABLE 4.3 – Résultats de surface post-implémentation sur FPGA Kintex-7.

	skewed alone			**skewed with lock**			Overhead	
	LUTs	FFs	BRAMs	LUTs	FFs	BRAMs	LUTs	FFs
↦ VARP-64	94	85	2	100	89	2	+ 6,38%	+ 4,71%
↦ Data Cache	3 301	1 297	18	3 308	1 302	18	+ 0,21%	+ 0,38%
↦ VARP-64	94	85	2	96	89	2	+ 2,13%	+ 4,71%
↦ Instr Cache	2 685	1 200	18	2 687	1 204	18	+ 0,07%	+ 0,33%
↦ Core	4 655	2 251	0	4 619	2 252	0	- 2,18%	+ 0,04%
CPU	10 657	5 006	36	10 630	5 017	36	- 0,25%	+ 0,18%

Le cache d'instructions ne supporte pas les instructions `lock` et `unlock` d'où la différence mineure avec 4 LUTs de moins en comparaison avec l'implémentation dans le cache de données. La capacité des mémoires BRAMs du composant est suffisante pour stocker les données et les métadonnées sans avoir besoin de rajouter un bloc supplémentaire.

4.2 Protection des mémoires avec l'introduction de TEE

La Section 4.1 présente une étude de mécanisme de sécurité au niveau des mémoires caches qui s'est effectuée dans le cadre du projet SCRATCHS, mais qui est en train d'être étendue avec un ingénieur pour avoir un démonstrateur plus complet avec un processeur capable de faire tourner un noyau Linux.

Dans la suite de mes travaux de recherche, je souhaite continuer dans ce type d'études de sécurité sur la microarchitecture. Du fait que mes activités passées et actuelles sont à la frontière entre le logiciel et le matériel, j'ai commencé à m'intéresser aux problématiques de sécurité des environnements d'exécution sécurisés (ou TEE). Un premier projet, financé par l'ANR, va me permettre d'étudier l'exploitation de ces mécanismes.

4.2.1 Problématique

Les travaux que j'initie autour de la sécurité et des TEEs se déroulent dans le cadre du projet ANR Secure-by-Design Computing Against Microarchitectural Attacks (SCAMA). Il s'agit d'un projet dans lequel nous souhaitons apporter des fonctions de sécurité dès la conception du circuit. Dans ce projet, nous sommes partis du constat que des attaques récentes telles que Spectre [99] ou Meltdown [100] ont montré que le matériel, qui est souvent considéré comme étant une couche qui exécute correctement les instructions, peut faire fuiter des informations sensibles relatives au logiciel. De plus, les architectures modernes peuvent ne pas être entièrement documentées : les éléments ou modules non documentés peuvent alors être exploités par un attaquant pour créer une attaque sur la microarchitecture.

Nous souhaitons apporter une réponse à la question suivante : "Quel niveau de sécurité doit-on introduire dans la conception d'un système sur puce tout en garantissant ses performances ?" Les réponses apportées doivent être :

- Pratiques : autrement dit, ne pas dégrader les performances.
- Extensibles : afin de pouvoir s'adapter à des attaques diverses sur la microarchitecture visée.
- Suffisamment génériques pour pouvoir s'adapter à différentes plateformes.

Les solutions apportées dans le projet vont impacter autant les couches matérielles que les couches logicielles d'une plateforme que nous développer avec un processeur RISC-V implémenté sur un composant reconfigurable FPGA. Le projet peut se décomposer en trois grandes étapes réalisées par trois des partenaires du projet.

Tout d'abord, une détection automatisée des attaques temporelles possibles sur les caches d'une plateforme implémentée dans le simulateur gem5 [1]. Cette étape, réalisée par des partenaires au laboratoire LTCI de Télécom Paris, devra permettre d'identifier des vulnérabilités sur des composants qui peuvent ne pas être documentés comme expliqué dans l'introduction de cette section. Dans une première contribution [101], le doctorant impliqué sur cette partie a déjà proposé un ensemble de simulations d'attaques visant des architectures x86 et ARM afin de montrer qu'il est possible d'identifier des suites d'instructions pouvant être utilisés pour automatiser la détection d'attaques.

Les autres étapes du projet consistent à réaliser des mécanismes de protection matériels et logiciels. Les mécanismes matériels seront réalisés par le laboratoire Hubert-Curien de Saint-Etienne : cela comprendra l'implémentation matérielle du modèle de détection développé par Télécom Paris ainsi que le développement de contre-mesures dont la nature exacte reste encore à déterminer.

Enfin, la contribution du Lab-STICC (entre l'ENSTA Brest et l'Université de Bretagne-Sud à Lorient), à laquelle je contribue, s'intéresse aux mécanismes de protection logiciels. Nous envisageons de mettre en place un TEE ainsi que l'utilisation de compilation à la volée (ou JIT) pour avoir des contre-mesures à la demande pour différentes parties du code exécutable. Le reste de la Section 4.2 donne quelques

4.2.2 Environnements d'exécution sécurisés pour architecture RISC-V

Les TEEs sont des environnements d'exécutions sécurisés qui permettent d'isoler des parties de logiciels appelées *enclaves*. Ces environnements sont disponibles sur plusieurs architectures : parmi les implémentations les plus connues, on peut citer SGX [102] et TDX [103] pour Intel et TrustZone [104] pour ARM. Bien qu'ayant pour but de protéger du logiciel, les TEEs se basent souvent sur des éléments matériels intégrés dans le processeur.

- Pour SGX, de nouvelles instructions et de nouveaux registres ont été ajoutés pour gérer le cycle de vie des enclaves ainsi que des modules matériels permettant de générer des signatures pour identifier les différentes enclaves.
- Pour TrustZone, on y trouve également de nouveaux registres ainsi qu'un bit supplémentaire (le NS bit, pour *Non-Secure* bit) qui permet de différencier les mondes sécurisé et non-sécurisé.

En ce qui concerne les TEEs compatibles avec l'architecture RISC-V, il en existe plusieurs parmi lesquelles Penglai [105], Keystone [30], Hex Five MultiZone [106]

1. https://www.gem5.org/

et Sanctum [107] (la Table 2 de l'article [108] de Munoz et al donne une vue plus exhaustive des solutions disponibles). Dans le cadre des travaux liés au projet ANR SCAMA, nous avons d'abord dû faire un premier état des lieux des TEEs existants et d'étudier certaines de leurs propriétés.

La Table 4.4 présente quelques TEEs existants visant une architecture RISC-V. Tout d'abord, en ce qui concerne la scalabilité des solutions proposées, le nombre d'enclaves est limité sauf pour TIMBER-V [109] et Penglai [105] : dans Sanctum, le nombre d'enclaves est lié au nombre de régions DRAM ; dans CURE, l'arbitre matériel limite à 13 régions possibles ; Keystone, dont les mécanismes sont basés sur les registres PMP du processeur, est par conséquent limité à un nombre d'enclaves égal au nombre de régions PMP (16 pour le processeur CVA6). Concernant les mécanismes de sécurité, deux solutions proposent des mécanismes de chiffrement et d'intégrité des enclaves : Penglai (inclus par défaut) et Keystone (en option).

TABLE 4.4 – Comparaison non exhaustive de TEEs pour l'architecture RISC-V (adapté de [105]).

Nom	**Scalabilité** Nombre d'enclaves	Granularité mémoire	**Sécurité** Chiffrement mémoire	Intégrité mémoire	**Implémentation**
Sanctum [107]	Régions DRAM	Région	✗	✗	Rocket
TIMBER-V [109]	Illimité	Page	✗	✗	Spike
Keystone [30]	PMPs	Région	On-chip	On-chip	QEmu et CVA6
CURE [110]	13	Région	✗	✗	Rocket
Penglai [105]	Illimité	Page	✓	✓	QEmu

Nous n'en sommes qu'au début du projet, nous n'avons pas encore défini les limites définitives du système qui sera développé d'ici à la fin du projet ANR SCAMA. Le choix du TEE à utiliser pour un premier démonstrateur dans le projet ANR SCAMA a été motivé par le fait d'avoir une implémentation sur carte FPGA. Dans cette optique, les solutions Sanctum, Keystone et CURE se sont révélées les plus intéressantes, car elles ont été validées sur des implémentations matérielles de processeur RISC-V. Le TEE Keystone a finalement été choisi, car il a non seulement été validée sur l'émulateur QEmu mais également sur l'implémentation CVA6 de l'OpenHW Group : les travaux actuels autour de la protection des caches présentés dans la Section 4.1 vont être réalisés sur cette même implémentation, on imagine donc une mise en commun de la plateforme entre ces deux projets qui permettrait de tirer profit de nos connaissances autour de ce processeur.

4.2.3 Solution envisagée

On envisage d'intégrer différentes contre-mesures dans les enclaves parmi lesquelles :

- **Exécution en temps constant :** Permet de s'assurer que les temps d'accès aux caches sont indépendants de la donnée secrète. Cette technique est utilisée par exemple dans des algorithmes de chiffrement Advanced Encryption Standard (AES) pour éviter une fuite de la clé en étudiant les variations temporelles [80].
- **Injection de bruit :** On introduit une part d'aléa dans les mesures de temps ou dans les accès aux données partagées. [111] présente une technique de ce type qui est utilisée dans des systèmes temps-réel et des systèmes embarqués sécurisés.
- **Déterminisme forcé :** L'objectif de cette technique est de se défaire des variations de temps qu'un attaquant pourraient exploiter. [112], [113] évoque ce mécanisme pour la sécurité des machines virtuelles et de l'informatique dans les nuages.
- **Partitionnement temporel :** La maîtrise des accès aux ressources partagées (d'un point de vue temporel). Cette approche est utilisée notamment dans des travaux implémentant du *cache flushing* [114] et des *execution leases* [115], un mécanisme qui permet de vérifier que les flux d'informations sont sûrs, et ce, jusqu'aux couches matérielles.
- **Partitionnement matériel :** Une isolation des couches matérielles pour s'assurer que chaque process à son propre espace dédié. On retrouve cela dans du *cache locking* [84], du *cache coloring* [116], ou du *quasi-partitioning* [117].

La suite de l'étude de cette solution se décomposera en plusieurs étapes :

- Une étude des différentes contre-mesures tant d'un point de vue sécurité que d'un point de vue performances. L'objectif est de définir le meilleur compromis entre la sécurité d'une enclave et ses performances. Nous étudierons également avec attention l'impact des mécanismes sur l'environnement logiciel (par exemple, nécessité d'avoir un compilateur spécifique) et sur la plateforme matérielle (registres supplémentaires ou modifications du pipeline du processeur).
- La définition d'un prototype avec plusieurs enclaves, chacune proposant une seule contremesure.
- Enfin, nous étudierons la possibilité de pouvoir changer le mécanisme de sécurité intégré à une enclave en cours d'exécution grâce à la compilation JIT, une technique que nous avons déjà étudié dans le cadre des travaux de la thèse de Quentin Ducasse présentés dans la Section 2.2.

4.3 Conclusion et perspectives

Les travaux présentés dans cette section est une synthèse des travaux de Nicolas Gaudin sur le projet Cominlabs SCRATCHS (publications liées présentes en annexe A.2) et des travaux de la thèse d'Oussama Elmnaouri débutée fin 2024 sur le projet ANR SCAMA (publications en annexe A.2).

Sauvegarde des verrous en cache

Dans la Section 4.1, nous avons proposé une solution hybride qui associe une technique de verrouillage de lignes de cache et un ajout d'aléa grâce à une politique de remplacement particulière. Cela permet d'ajouter une couche de sécurité supplémentaire au niveau de la micro-architecture des caches dans un processeur embarqué RISC-V. D'autres travaux sont en cours afin de porter ces mécanismes sur un processeur CVA6 : un processeur RISC-V 64 bits capable de faire tourner un noyau Linux ; le postdoctorant impliqué sur cette extension du projet SCRATCHS devrait également étudier l'implémentation d'un module de sauvegarde des verrous en cache qui viendrait résoudre le problème d'invalidation totale du cache lors du renouvellement de clé de l'IDF.

Travaux autour des TEEs

Les travaux de la thèse d'Oussama Elmnaouri présentés dans la Section 4.2 ont démarré il y a environ un an à l'heure de l'écriture de ce manuscrit. Dans le cadre de la thèse, nous sommes en plus en train d'étudier une contribution qui serait un compromis entre les approches HybCache [118] et "Composable cachelets" [119]. HybCache propose une architecture avec un partitionnement statique des voies où le cache aurait des contremesures pour un code s'exécutant dans un domaine isolé, contrairement à des exécutions non isolées où le cache se comporterait normalement. L'approche présentée dans [119] propose, quant à elle, un partitionnement sous forme de "cachelet" (un sous-ensemble de lignes) pour chaque enclave. Les deux solutions ont été étudiées pour une architecture x86, même si HybCache est indépendant de l'architecture. On souhaiterait étudier une solution qui associe un partitionnement dynamique à un grain fin tout en conservant une isolation forte pour le code de chaque enclave.

D'autre part, un postdoc est en cours de recrutement pour proposer des mécanismes matériels qui permettront d'améliorer la sécurité des attaques par canaux cachés sur des TEEs pour l'architecture RISC-V. Nous avons déjà identifié quelques travaux existants [120], [121] et nous aimerions aller plus loin dans le cadre de ce postdoc. Cela me permettrait à la fois d'approfondir mes connaissances dans le domaine des TEEs tout en collaborant avec mon doctorant.

Chapitre 5

Conclusion et perspectives

5.1 Conclusion

Dans ce manuscrit, j'ai présenté une synthèse des différents travaux autour de la sécurité dans les systèmes embarqués que j'ai pu aborder pendant mes années d'enseignant-chercheur à CentraleSupélec puis à l'ENSTA. Dans ce document, il me semblait plus pertinent de ne pas procéder par ordre chronologique, mais plutôt avec un classement thématique. Les activités menées dans ma thèse de doctorat étaient plus focalisées sur l'implémentation de mécanismes de sécurité dans des composants reconfigurables FPGA. Dès mes débuts à CentraleSupélec, les travaux ont commencé à associer des contributions sur la couche matérielle et d'autres sur la couche logicielle.

Le manuscrit a montré que le logiciel et le matériel sont aujourd'hui indissociables lorsque l'on s'intéresse à la sécurité des systèmes embarqués et que cette thématique peut s'aborder à différents niveaux :

- Haut-niveau (voir chapitre 3) dans la mesure où les travaux abordent l'implémentation et l'étude de blocs matériels pour des problématiques de sécurité au sein d'une passerelle sécurisée ou en support d'une intrusion de détection dans une meute de drones.
- À un niveau que je qualifierais d'intermédiaire comme cela a été étudié dans le chapitre 2 où les interactions entre le matériel et le logiciel sont les plus fortes.
- Enfin, dans le chapitre 4, des travaux au niveau de la micro-architecture et plus particulièrement autour des mémoires cache ont été abordés.

Certains travaux sont toujours en cours et vont m'amener à approfondir certains sujets dans les années à venir, ces sujets sont détaillés dans les perspectives (Section 5.2).

5.2 Perspectives

Toujours dans un objectif de sécurisation des codes sur du matériel embarqué, il existe des TEEs similaires à TrustZone mais pour architecture RISC-V (par exemple, Keystone[1] ou HexFive Multizone[2]). Les travaux possibles autour de ces environnements d'exécution sécurisés sont multiples :

— Concernant le côté matériel, certains processeurs tels que le CV32E40S[3] de l'OpenHwGroup ont un support partiel de ce type de TEE. Il serait intéressant de développer une preuve de concept d'environnement sécurisé sur ce processeur embarqué. Cette plateforme pourrait alors servir de base à des travaux complémentaires tant sur le volet logiciel que matériel.

— Dans la thèse de Quentin Ducasse, nous avons étudié une contre-mesure de sécurité à deux "domaines" (ou niveaux de sécurité, de manière similaire à une TEE ARM TrustZone). Les TEEs type Keystone étant open-source, j'aimerais étudier la possibilité de pouvoir faire une protection de code "multi-domaine" ce qui pousserait à étudier à la fois des aspects sécurité, mais également l'optimisation de ce mécanisme sur un composant reconfigurable (utilisation des mémoires, résilience des paramètres de configuration de ce TEE, temps de traitement).

Concernant la sécurité sur la micro-architecture des processeurs, nous avons récemment obtenu une extension du projet Cominlabs SCRATCHS qui nous permettra d'embaucher un ingénieur pendant un an afin d'arriver à un démonstrateur complet de la plateforme contenant des contributions des deux thèses du projet. Les travaux menés dans ce projet vont avoir une action double à moyen terme :

— La plateforme développée dans SCRATCHS devrait être réutilisée comme base pour le projet ANR SCAMA qui a démarré récemment. Il s'agit d'un projet qui inclut les laboratoires LTCI à Paris (coordinateur du projet), Hubert-Curien à Saint-Étienne et LIRMM à Montpellier. Ce projet a pour objectif de proposer des mécanismes *hybrides* (logiciels et matériels) contre les vulnérabilités sur la micro-architecture d'un processeur RISC-V, les vulnérabilités ayant été remontées dans un premier par des outils d'apprentissage profond.

— Cette plateforme pourra de même ensuite être utilisé pour des extensions sur le projet : pour l'instant, on y utilise un processeur embarqué. On pourrait être amené à étudier quelles seraient les fuites dans un composant multicoeurs et multitâches, toujours basé sur l'architecture RISC-V qui

1. https://keystone-enclave.org/
2. https://hex-five.com/multizone-security-tee-riscv/
3. https://github.com/openhwgroup/cv32e40s

nous permet d'avoir des outils pour étudier finement les problématiques de sécurité par canaux cachés.

Dans l'objectif d'approfondir mes connaissances sur les attaques sur la micro-architecture, je souhaite également étudier les possibilités d'attaques sur des systèmes d'enclaves et de TEE [122]-[124] avec des contre-mesures qui seraient potentiellement matérielles avec du logiciel bas-niveau.

Dans l'article BUSted [125], une équipe de l'université de Minho au Portugal a proposé d'analyser les variations temporelles de la logique d'arbitrage du bus de données sur des familles de microcontrôleurs pour contourner les mécanismes de protection de la mémoire. Ces travaux ont été menés principalement sur des architectures ARM, un composant RISC-V (GigaDevice GD32VF103) a également été analysé. Une étude au croisement de BUSted [125] et les travaux de Zonta et al. [71] pourrait mettre en valeur des failles sur la micro-architecture de certaines architectures RISC-V implémentées sur un composant reconfigurable FPGA.

Un troisième volet de mes activités de recherche sera un peu plus haut niveau où j'étudierais des applications de sécurité où un composant matériel peut être utile. C'est le cas de la thèse qu'on vient de démarrer dans la Chaire de Cyber-défense des systèmes navals : dans une flotte de drones (marins et terrestres), nous souhaitons étudier comme les algorithmes d'apprentissage peuvent être utilisés pour des sondes de détection d'intrusion IDS. Dans ce cadre applicatif, il est intéressant de vouloir accélérer les calculs et d'implémenter cela sur une architecture embarquée sur les drones. Lorsque nous aurons un premier démonstrateur avec une détection d'intrusion fonctionnelle, un des objectifs est de s'intéresser des "poisoning attacks" [126], [127] et de voir s'il serait possible d'intégrer des contremesures dans le matériel pour mieux résister à ce type d'attaques.

Nous avons initié une collaboration avec Naval Group dans le cadre de la chaire de cyberdéfense des systèmes navals. De plus, Thales est une autre entreprise importante dans le monde de la sécurité embarquée et en particulier dans l'écosystème OpenHwGroup. Dans les futurs projets, je compte renforcer mes collaborations avec des industriels et des organismes de défense. De même, bien que mes collaborations à l'international soient peu nombreuses pour l'instant, je chercherais à les construire dans les années à venir.

Toutes ces activités de recherche seront supports de publications dans les revues et les conférences relatives aux GDR Sécurité Informatique et GDR SoC^2 (les architectures reconfigurables font partie de ce dernier). En plus des publications, je prêterais un oeil attentif au partage des codes qui permettent d'obtenir les résultats : ce sont des aspects que j'ai expérimentés dans mon activité de relecteur d'artefact et dans la thèse de Quentin Ducasse (évoquée plus tôt dans cette section) avec lequel nous avons diffusé plusieurs outils relatifs à ses travaux [4] [5].

4. https://github.com/QDucasse/jitdomain-tests
5. https://github.com/QDucasse/gigue

Liste d'abréviations

Annexes

Annexe A

Informations complémentaires

A.1 CV résumé

Expériences professionnelles

2019 - ... Enseignant-chercheur à l'ENSTA Bretagne (désormais ENSTA, campus de Brest) et au laboratoire Lab-STICC (Brest, France).

2017 - 2019 Ingénieur en logiciel embarqué chez Thales (Cholet, France).

2014 - 2017 Enseignant-chercheur à CentraleSupélec, campus de Rennes et à l'IETR (Rennes, France).

Diplômes

2012 Thèse de doctorat (Université de Bretagne-Sud, Lorient, France).

2009 Diplôme d'ingénieur spécialité optoélectronique (Télécom Saint-Étienne, Saint-Étienne, France).

2008 Bachelor of Engineering in Electronic Engineering (University of South Wales, Royaume-Uni).

A.2 Liste des thèses co-encadrées

Thèses en cours

1. **11/2024 -** ... **Oussama Elmnaouri (Projet ANR SCAMA)**
   - Software Mitigations for Cache and Covert Timing SCAs. Lab-STICC, ENSTA. Encadrée à hauteur de 30%.
   - Directeur de thèse : Loïc Lagadec (Professeur, ENSTA). Co-encadrants : Vianney Lapôtre (Université de Bretagne-Sud).
   - Publications associées : [128], [129].
2. **11/2023 -** ... **Pierre Garreau (Chaire de Cyberdéfense des Systèmes Navals)**
   - Embedded IDS for RISC-V targeting drone systems. Lab-STICC, ENSTA. Encadrée à hauteur de 25%.

- Directeur de thèse : Loïc Lagadec (Professeur, ENSTA). Co-encadrants : Jean-Christophe Cexus (Professeur, ENSTA) et Julien Francq (Naval Group).
- Publications associées : [130], [131].

3. **2022 - ... Aya Jendoubi (ANR TrustGW)**
   - Enhancing Security in Heterogeneous Virtualized Systems : A Focus on I/O Attacks in the existence of IOMMU in a RISC-V architecture. IETR, INSA Rennes. Encadrée à hauteur de 20%.
   - Directeur de thèse : Jean-Christophe Prévotet (professeur, INSA Rennes). Co-encadrant : Philippe Tanguy (Maître de Conférences, Université de Bretagne-Sud).
   - Publications associées : [132], [133].

Thèses soutenues

1. **2021 - 2024. Nicolas Gaudin (Cominlabs SCRATCHS)**
   - Security mechanisms against timing cache attacks. Lab-STICC, Université de Bretagne-Sud. Encadrée à hauteur de 40%.
   - Directeur de thèse : Guy Gogniat (Université de Bretagne-Sud). Co-encadrant : Vianney Lapôtre (Université de Bretagne-Sud).
   - Publications associées : [98], [134]-[136].
2. **2021 - 2024. Quentin Ducasse (Brest Métropole et Pôle d'Excellence Cyber)**
   - Low-level JIT security mechanisms for VMs running on RISC-V processors. Lab-STICC, ENSTA Bretagne. Encadrée à hauteur de 50%.
   - Directeur de thèse : Loïc Lagadec (Professeur, ENSTA).
   - Publications associées : [33], [137]-[140].
   - Publication en cours de révision : [39].
3. **2015 - 2018. Muhammad Abdul Wahab (Cominlabs HardBlare)**
   - Hardware support for the security analysis of embedded softwares : applications on information flow control and malware analysis. IETR, CentraleSupélec Rennes. Encadrée à hauteur de 50%.
   - Directeur de thèse : Christophe Moy (Professeur, Université de Rennes). Co-encadrant : Guillaume Hiet (INRIA Rennes).
   - Publications associées : [141]-[152].
4. **2015 - 2020. Mounir Nasr Allah (Cominlabs HardBlare)**

- HardBlare - Information flow tracking through static and dynamic analysis. INRIA, CentraleSupélec Rennes. Encadrée à hauteur de 20%.
- Directeur de thèse : Ludovic Mé (Professeur, INRIA Rennes). Co-encadrant : Guillaume Hiet (INRIA Rennes).

A.3 Jurys et expertises

CSI

1. Mahreen Khan (LabSoC, Télécom Paris). Analyse des architectures matérielles de type RISC-V, défenses contre certains canaux cachés, notamment les attaques via les mémoires caches. Mahreen Khan a commencé sa thèse en 2024, son deuxième CSI a eu lieu début juin 2025.

Expertise de projets

J'ai été expert externe pour l'évaluation de 2 projets ANR en 2022 et 2024.

Participation aux comités de programme

J'ai participé à 18 comités de programmes parmi lesquels :

- 2026 : NDSS.
- 2024, 2023, 2022, 2021, 2020, 2019 : LASCAS.
- 2024, 2023, 2022 : Baltic Electronics Conference.
- 2024 : ACSAC.
- 2024 : IEEE CSR HACS.

Une autre fonction est la participation en tant que *Artifact Evaluation Committee* (AEC). Ce rôle est relativement nouveau : une fois les papiers acceptés, les auteurs sont invités à soumettre des artefacts qui doivent permettre de reproduire les résultats. Au-delà de la pure production du code, une part importante de la notation est également accordée à la documentation qu'on retrouve le plus souvent dans un dépôt mis à disposition par les auteurs. Certaines conférences renommées dans le domaine de la sécurité ont d'ailleurs mis en place le site "Security Research Artifacts" qui explique ce qui est attendu dans ces soumissions d'artefacts.

Je suis membre d'AEC dans les conférences de sécurité suivantes :

- 2025, 2024, 2023 : Usenix. J'ai d'ailleurs obtenu un "Distinguished Reviewer Awards" pour l'édition 2024.
- 2025, 2024, 2023 : CCS.
- 2024 : ACSAC.

A.4 Responsabilités scientifiques

Financements nationaux

1. ANR SCAMA (2024 à 2028).
    - Subvention de 200.000 euros environ. Cette subvention permet de financer une thèse complète ainsi que les coûts associés. La thèse va démarrer en novembre 2024.
    - Responsable d'un lot de travail.
    - Responsable du projet par intérim (indisponibilité de la porteuse pendant 6 mois).
2. ANR TrustGW (2022 à 2025).
    - Subvention pour amortir les déplacements liés au projet.
    - Responsable d'un lot de travail sur la mise en œuvre du démonstrateur final.
    - Co-encadrement de la thèse d'Aya Jendoubi (IETR Rennes).

Financements collectivités locales (région, métropole de Brest, Pôle d'Excellence Cyber)

1. Chaire de cyberdéfense des Systèmes Navals (2023 à 2026). Subvention pour la thèse complète de Pierre Garreau, avec un co-encadrement Naval Group.
2. Labex Cominlabs SCRATCHS (2021 à 2025). Subvention pour le financement d'une thèse (Nicolas Gaudin). Co-encadrement avec Guy Gogniat et Vianney Lapôtre (Lab-STICC).
3. Demi-financement Brest Métropole et demi-financement par le Pôle d'Excellence Cyber (2021 à 2024). Subvention pour la thèse de Quentin Ducasse.
4. Pôle d'Excellence Cyber (2017).

- Subvention pour une thèse sur le thème “Conception et réalisation de solutions matérielles/logicielles sur plateformes SoC de crypto-systèmes basés chaos pour des objets connectés sécurisés” co-encadrée avec Olivier Desforges (INSA Rennes) et Safwan El Assad (Polytech Nantes).
- Encadrement de la thèse abandonné suite à mon départ dans le privé en 2017.

5. Labex Cominlabs HardBlare (2015 à 2020).
   - Subvention pour le financement de deux thèses.
   - Co-encadrement des deux thèses avec Guillaume Hiet (INRIA CIDRE).

Financements sans candidats

Je note dans cette partie les financements obtenus, mais qui n'avaient malheureusement pas trouvé de candidats dans les délais impartis.

1. PEPR ARSENE (2023). Financement complet d'une thèse dirigée par Arnaud Tisserand dans laquelle j'aurais été co-encadrant. Le financement de cette thèse devrait être transformé en financement d'un postdoc pour travailler sur un sujet équivalent.

A.5 Responsabilités administratives

Responsabilités en cours et futures

2024 - ... Correspondant ENSTA pour le parcours de formations en Cybersécurité CyberSkills4All. Il s'agit d'un des projets lauréats du programme “Compétences et métiers d'avenir” de France 2030 : le programme démarre à peine, on sait qu'il y aura deux micro-formations à monter à l'ENSTA Brest.

2024 - ... Correspondant STIC (*Sciences et Technologies de l'Information et de la Communication*) pour un cursus spécialisé “défense et sécurité” à destination des élèves militaires de l'établissement qui résulte de la fusion ENSTA Bretagne / ENSTA Paris (diplôme co-accrédité avec l'ISAE-Supaero).

2022 - ... Responsable de la filière par apprentissage en “systèmes embarqués” à l'ENSTA.

Responsabilités passées

2020-2021 Coresponsable du mastère spécialisé “Cybersécurité des systèmes maritimes et portuaires” avec l’IMT Atlantique.

A.6 Synthèse des enseignements

Volume horaire

Le tableau A.1 présente une synthèse de mes charges d’enseignements année par année. L’ENSTA Bretagne et l’ENSTA Paris ayant fusionné début 2025 pour former une seule ENSTA, le nom de l’établissement a été modifié en conséquence.

Table A.1 – Synthèse des charges d’enseignements.

Année	Établissement	Charge (h ETD)	Niveau
2014/2015	CentraleSupélec Rennes	192	L3 à M2
2015/2016	CentraleSupélec Rennes	192	L3 à M2
2016/2017	CentraleSupélec Rennes	192	L3 à M2
2019/2020	ENSTA Bretagne	200	L3 à M2
2020/2021	ENSTA Bretagne	213	L3 à M2
2021/2022	ENSTA Bretagne	310	L3 à M2
2022/2023	ENSTA Bretagne	316	L3 à M2
2023/2024	ENSTA Bretagne	310	L3 à M2
2024/2025	ENSTA	214	L3 à M2

Vue par matière

Table A.2 – Synthèse par matière.

Année	Niveau	Diplôme	Intitulé	Nature	Volume horaire annuel
2019/2025	L3	FIPA	Introduction aux réseaux	CM - TD - TP	30
2020/2025	M1	FIPA	C embarqué	CM - TD	30
2019/2025	L3	FISE	Introduction aux réseaux	CM - TD - TP	30
2022/2025	M2	FIPA	OS embarqué et temps réel	CM - TP	30
2022/2025	M2	FIPA	Calcul intensif embarqué	CM - TD	30
2019/2025	L3	FISE	Introduction à la programmation	TD	12
2019/2025	M2	FISE	Sécurité appliquée aux réseaux	CM - TD - TP	30

Dans le tableau A.2, j'ai mis uniquement les enseignements relatifs à l'ENSTA. FIPA est le diplôme par apprentissage spécialisé en systèmes embarqués ; FISE est le diplôme d'ingénieur généraliste, j'intervenais essentiellement dans la voie d'approfondissement CSN (Conception de Systèmes Numériques). Les volumes horaires indiqués sont des heures réelles et les volumes sont ceux de la dernière année d'enseignement (ceux-ci ont pu varier d'1 ou 2 séances d'une année sur l'autre). Ce tableau ne contient pas les charges annexes telles que la surveillance d'examen ou les décharges octroyées aux responsables de filière (ce qui est mon cas avec la filière par apprentissage en systèmes embarqués à l'ENSTA).

Bibliographie

[1] Nickolai Zeldovich, “Securing Untrustworthy Software Using Information Flow Control,” thèse de doct., Stanford University, 2008. adresse : https://dl.acm.org/doi/10.5555/1369312 (cité page 6).

[2] Jacob Zimmermann, Ludovic Mé, Christophe Bidan, “Introducing Reference Flow Control for Detecting Intrusion Symptoms at the OS Level,” in *Recent Advances in Intrusion Detection*, A. Wespi, G. Vigna et L. Deri, éd., Berlin, Heidelberg : Springer Berlin Heidelberg, oct. 2002, p. 292-306, ISBN : 978-3-540-36084-1 (cité page 6).

[3] Cheng Wang, Shiliang Hu, Ho-seop Kim, “StarDBT : An Efficient Multi-platform Dynamic Binary Translation System,” in *Advances in Computer Systems Architecture*, L. Choi, Y. Paek et S. Cho, éd., Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 4-15, ISBN : 978-3-540-74309-5 (cité page 7).

[4] Chi-Keung Luk, Robert Cohn, Robert Muth, “Pin : building customized program analysis tools with dynamic instrumentation,” *SIGPLAN Not.*, t. 40, nº 6, p. 190-200, juin 2005, ISSN : 0362-1340. DOI : 10.1145/1064978.1065034. adresse : https://doi.org/10.1145/1064978.1065034 (cité page 7).

[5] Michael Dalton, Hari Kannan, Christos Kozyrakis, “Raksha : a flexible information flow architecture for software security,” *SIGARCH Comput. Archit. News*, t. 35, nº 2, p. 482-493, juin 2007, ISSN : 0163-5964. DOI : 10.1145/1273440.1250722. adresse : https://doi.org/10.1145/1273440.1250722 (cité pages 7-9).

[6] André DeHon, Ben Karel, Thomas F. Knight, “Preliminary design of the SAFE platform,” in *Proceedings of the 6th Workshop on Programming Languages and Operating Systems*, sér. PLOS ’11, Cascais, Portugal : Association for Computing Machinery, oct. 2011, ISBN : 9781450309790. DOI : 10.1145/2039239.2039245. adresse : https://doi.org/10.1145/2039239.2039245 (cité page 7).

[7] Hari Kannan, Michael Dalton, Christos Kozyrakis, “Decoupling Dynamic Information Flow Tracking with a dedicated coprocessor,” in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, juin 2009, p. 105-114. DOI : 10.1109/DSN.2009.5270347 (cité pages 7-9, 15).

[8] Ingoo Heo, Minsu Kim, Yongje Lee, "Implementing an Application-Specific Instruction-Set Processor for System-Level Dynamic Program Analysis Engines," *ACM Trans. Des. Autom. Electron. Syst.*, t. 20, nº 4, sept. 2015, ISSN : 1084-4309. DOI : 10.1145/2746238. adresse : https://doi.org/10.1145/2746238 (cité pages 9, 14, 15).

[9] Daniel Y. Deng, Daniel Lo, Greg Malysa, Skyler Schneider, G. Edward Suh, "Flexible and Efficient Instruction-Grained Run-Time Monitoring Using On-Chip Reconfigurable Fabric," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, déc. 2010, p. 137-148. DOI : 10.1109/MICRO.2010.17 (cité pages 9, 15).

[10] Daniel Y. Deng, G. Edward Suh, "High-performance parallel accelerator for flexible and efficient run-time monitoring," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, juin 2012, p. 1-12. DOI : 10.1109/DSN.2012.6263925 (cité page 9).

[11] Jinyong Lee, Ingoo Heo, Yongje Lee, Yunheung Paek, "Efficient Security Monitoring with the Core Debug Interface in an Embedded Processor," *ACM Trans. Des. Autom. Electron. Syst.*, t. 22, nº 1, mai 2016, ISSN : 1084-4309. DOI : 10.1145/2907611. adresse : https://doi.org/10.1145/2907611 (cité page 9).

[12] Udit Dhawan, Catalin Hritcu, Raphael Rubin, "Architectural Support for Software-Defined Metadata Processing," *SIGARCH Comput. Archit. News*, t. 43, nº 1, p. 487-502, mars 2015, ISSN : 0163-5964. DOI : 10.1145/2786763.2694383. adresse : https://doi.org/10.1145/2786763.2694383 (cité page 9).

[13] Olatunji Ruwase, Phillip B. Gibbons, Todd C. Mowry, "Parallelizing dynamic information flow tracking," in *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, sér. SPAA '08, Munich, Germany : Association for Computing Machinery, juin 2008, p. 35-45, ISBN : 9781595939739. DOI : 10.1145/1378533.1378538. adresse : https://doi.org/10.1145/1378533.1378538 (cité page 9).

[14] Shimin Chen, Michael Kozuch, Theodoros Strigkos, "Flexible Hardware Acceleration for Instruction-Grain Program Monitoring," *SIGARCH Comput. Archit. News*, t. 36, nº 3, p. 377-388, juin 2008, ISSN : 0163-5964. DOI : 10.1145/1394608.1382153. adresse : https://doi.org/10.1145/1394608.1382153 (cité page 9).

[15] Lucas Davi, Matthias Hanreich, Debayan Paul, "HAFIX : hardware-assisted flow integrity extension," in *Proceedings of the 52nd Annual Design Automation Conference*, sér. DAC '15, San Francisco, California : Association for Computing Machinery, juin 2015, ISBN : 9781450335201.

DOI : 10.1145/2744769.2744847. adresse : https://doi.org/10.1145/2744769.2744847 (cité page 9).

[16] Olle Svanfeldt-Winter, Sebastien Lafond, Johan Lilius, *Evaluation of the Energy Efficiency of ARM Based Processors for Cloud Infrastructure*, Odefinierat/okänt. Turku Centre for Computer Science (TUCS), 2010, TUCS Technical Report No 991, December 2010, ISBN : 978-952-12-2516-1 (cité page 14).

[17] Jim Smith, Ravi Nair, *Virtual machines : versatile platforms for systems and processes*. Elsevier, 2005 (cité page 15).

[18] Taemin Park, Karel Dhondt, David Gens, Yeoul Na, Stijn Volckaert, Michael Franz, "NoJITsu : locking down JavaScript engines," in *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS'20)*, San Diego, California, USA : The Internet Society, fév. 2020. DOI : 10.14722/ndss.2020.24262 (cité pages 16, 17).

[19] Aleph One, "Smashing the stack for fun and profit," *Phrack magazine*, t. 7, nº 49, 1996. adresse : http://phrack.org/issues/49/14.html (cité page 16).

[20] Hovav Shacham, "The geometry of innocent flesh on the bone : return-into-libc without function calls (on the x86)," in *Proceedings of the 14th ACM SIGSAC Conference on Computer and Communications Security (CCS'07)*, Alexandria, Virginia, USA : ACM, oct. 2007, p. 552-561. DOI : 10.1145/1315245.1315313 (cité page 16).

[21] Hong Hu, Shweta Shinde, Sendroiu Adrian, Zheng Leong Chua, Prateek Saxena, Zhenkai Liang, "Data-oriented programming : on the expressiveness of non-control data attacks," in *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P'16)*, San Jose, California, USA : IEEE, août 2016, p. 969-986. DOI : 10.1109/sp.2016.62 (cité page 17).

[22] Dionysus Blazakis, "Interpreter Exploitation," in *Proceedings of the 4th USENIX Workshop on Offensive Technologies (WOOT'10)*, Washington DC, USA : USENIX Association, août 2010. adresse : https://dl.acm.org/doi/10.5555/1925004.1925011 (cité page 17).

[23] Kevin Z Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, Ahmad-Reza Sadeghi, "Just-in-time code reuse : on the effectiveness of fine-grained address space layout randomization," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P'13)*, Berkeley, California, USA : IEEE, mai 2013, p. 574-588. DOI : 10.1109/SP.2013.45 (cité page 17).

[24] Wilson Lian, Hovav Shacham, Stefan Savage, "A call to ARMs : understanding the costs and benefits of JIT spraying mitigations," in *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS'17)*, San Diego, California, USA : The Internet Society, fév. 2017. DOI : 10.14722/ndss.2017.23108 (cité page 17).

[25] Team PaX, "PaX team objectives," 2003. adresse : http://pax.grsecurity.net/docs/pax.txt (cité page 17).

[26] Team PaX, "PaX Non-Executable pages design (NOEXEC)," 2003. adresse : http://pax.grsecurity.net/docs/aslr.txt (cité page 17).

[27] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, Jay Ligatti, "Control-flow integrity principles, implementations, and applications," *TISSEC : ACM Transactions on Information and System Security*, t. 13, nº 1, p. 1-40, nov. 2009. DOI : 10.1145/1609956.1609960 (cité page 17).

[28] ARM, *Learn the architecture - TrustZone for AArch64, Version 1.1*, 2021. adresse : https://developer.arm.com/documentation/102418/0101/ (cité page 17).

[29] Victor Costan, Srinivas Devadas, "Intel SGX explained," *Cryptology ePrint Archive (IACR)*, jan. 2016. adresse : https://eprint.iacr.org/2016/086 (cité page 17).

[30] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, Dawn Song, "Keystone : An Open Framework for Architecting Trusted Execution Environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, sér. EuroSys '20, avr. 2020 (cité pages 17, 53, 54).

[31] Soyeon Park, Sangho Lee, Wen Xu, Hyungon Moon, Taesoo Kim, "libmpk : software abstraction for Intel Memory Protection Keys (Intel MPK)," in *Proceedings of the USENIX Annual Technical Conference (ATC'19)*, Renton, Washington, USA : USENIX Association, juill. 2019, p. 241-254. adresse : https://www.usenix.org/conference/atc19/presentation/park-soyeon (cité page 17).

[32] Guillermo Polito, Pablo Tesone, Stéphane Ducasse, "Cross-ISA testing of the Pharo VM : lessons learned while porting to ARMv8," in *Proceedings of the 18th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes (MPLR'21)*, ACM, sept. 2021, p. 16-25. DOI : 10.1145/3475738.3480715 (cité page 17).

[33] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "Gigue : A JIT Code Binary Generator for Hardware Testing," in *2023 Workshop on Virtual Machines and Language Implementations*, oct. 2023 (cité pages 18, 68).

[34] Haeyoung Kim, Jinjae Lee, Derry Pratama, Asep Muhamad Awaludin, Howon Kim, Donghyun Kwon, “RIMI : instruction-level memory isolation for embedded systems on RISC-V,” in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD’20)*, San Diego, California, USA : ACM, nov. 2020, p. 1-9. DOI : 10.1145/3400302.3415727 (cité page 21).

[35] Haeyoung Kim, Harashta Tatimma Larasati, Jonguk Park, Howon Kim, Donghyun Kwon, “DEMIX : Domain-Enforced Memory Isolation for Embedded System,” *Sensors*, t. 23, nº 7, p. 3568, mars 2023. DOI : 10.3390/s23073568 (cité page 21).

[36] Yu Wang, Jinting Wu, Haodong Zheng, Zhenyu Ning, Boyuan He, Fengwei Zhang, “Raft : Hardware-assisted Dynamic Information Flow Tracking for Runtime Protection on RISC-V,” in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, sér. RAID ’23, Hong Kong, China : Association for Computing Machinery, oct. 2023, p. 595-608, ISBN : 9798400707650. DOI : 10.1145/3607199.3607246. adresse : https://doi.org/10.1145/3607199.3607246 (cité page 25).

[37] William Pensec, “Enhanced Processor Defence Against Physical and Software Threats by Securing DIFT Against Fault Injection Attacks,” Theses, Université Bretagne sud, déc. 2024. adresse : https://hal.science/tel-04862037 (cité page 25).

[38] RISC-V International / riscv-non-isa, *RISC-V Processor Trace Specification*, https://github.com/riscv-non-isa/riscv-trace-spec, GitHub repository, latest release version 2.0 (tag : 2.0-20250616), released June 23, 2025, juin 2025 (cité page 26).

[39] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, “JITDomain : Instruction-Level JIT Code Isolation,” *Microprocessors and Microsystems*, ELSEVIER, éd., août 2025, article en révision mineure (cité pages 26, 68).

[40] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, Biplab Sikdar, “A Survey on IoT Security : Application Areas, Security Threats, and Solution Architectures,” *IEEE Access*, t. 7, p. 82 721-82 743, juin 2019. DOI : 10.1109/ACCESS.2019.2924045 (cité page 27).

[41] Daniele Sgandurra, Emil Lupu, “Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems,” *ACM Comput. Surv.*, t. 48, nº 3, fév. 2016, ISSN : 0360-0300. DOI : 10.1145/2856126. adresse : https://doi.org/10.1145/2856126 (cité page 28).

[42] Tianxu Li, Mohamed El-Bouazzati, Camille Monière, Philippe Tanguy, Guy Gogniat, "Comparison Between In-Core Hardware IDS, Off-Core Hardware IDS and Software IDS," in *Design and Architecture for Signal and Image Processing*, J. Lorandel et A. Kamaleldin, éd., Cham : Springer Nature Switzerland, avr. 2025, p. 108-120, ISBN : 978-3-031-87897-8 (cité page 28).

[43] Rieul Ducousso, "Securing access to and from peripherals in a multicore RISC-V architecture used for virtualization," thèse de doct., Sorbonne University, mars 2023 (cité page 30).

[44] Thore Tiemann, Zane Weissman, Thomas Eisenbarth, Berk Sunar, "IOTLB-SC : An Accelerator-Independent Leakage Source in Modern Cloud Systems," in *Proceedings of the ACM Asia Conference on Computer and Communications Security*, sér. ASIA CCS '23, ACM, juill. 2023. DOI : 10.1145/3579856.3582838. adresse : http://dx.doi.org/10.1145/3579856.3582838 (cité page 30).

[45] Ahmad Atamli, Giuseppe Petracca, Jon Crowcroft, "IO-Trust : An out-of-band trusted memory acquisition for intrusion detection and Forensics investigations in cloud IOMMU based systems," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, New York, NY, USA : Association for Computing Machinery, août 2019, ISBN : 9781450371643. DOI : 10.1145/3339252.3340511. adresse : https://doi.org/10.1145/3339252.3340511 (cité page 31).

[46] Christoph Peglow, "Security analysis of hybrid Intel CPU/FPGA platforms using IOMMUs against I/O attacks," Master's Thesis, University of Lübeck, Lübeck, Germany, juill. 2020 (cité page 31).

[47] Emre Karabulut, Amro Awad, Aydin Aysu, "SS-AXI : Secure and Safe Access Control Mechanism for Multi-Tenant Cloud FPGAs," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, mai 2023, p. 1-5. DOI : 10.1109/ISCAS46773.2023.10181609 (cité page 31).

[48] Joel Mandebi Mbongue, Sujan Kumar Saha, Christophe Bobda, "A Security Architecture for Domain Isolation in Multi-Tenant Cloud FPGAs," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, juill. 2021, p. 290-295. DOI : 10.1109/ISVLSI51109.2021.00060 (cité page 31).

[49] Rana Elnaggar, Ramesh Karri, Krishnendu Chakrabarty, "Multi-Tenant FPGA-based Reconfigurable Systems : Attacks and Defenses," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, mars 2019, p. 7-12. DOI : 10.23919/DATE.2019.8714904 (cité page 31).

[50] Benoît Morgan, Eric Alata, Vincent Nicomette, Mohamed Kaaniche, "IOMMU protection against I/O attacks : A vulnerability and a proof-of-concept," *Journal of the Brazilian Computer Society*, t. 24, déc. 2018. DOI : 10.1186/s13173-017-0066-7 (cité page 31).

[51] Lilian Bossuet, El Mehdi Benhani, "Security Assessment of Heterogeneous SoC-FPGA : On the Practicality of Cache Timing Attacks," in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, oct. 2021, p. 1-6. DOI : 10.1109/VLSI-SoC53125.2021.9607012 (cité page 31).

[52] El Mehdi Benhani, Cedric Marchand, Alain Aubert, Lilian Bossuet, "On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, sept. 2017, p. 108-113. DOI : 10.1109/SOCC.2017.8226018 (cité page 31).

[53] José Martins, Adriano Tavares, Marco Solieri, Marko Bertogna, Sandro Pinto, "Bao : A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems," in *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, M. Bertogna et F. Terraneo, éd., sér. Open Access Series in Informatics (OASIcs), t. 77, Dagstuhl, Germany : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, jan. 2020, 3 :1-3 :14, ISBN : 978-3-95977-136-8. DOI : 10.4230/OASIcs.NG-RES.2020.3. adresse : https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.NG-RES.2020.3 (cité page 34).

[54] Bruno Sá, Francisco Marques, Manuel Rodriguez, José Martins, Sandro Pinto, "Holistic RISC-V Virtualization : CVA6-based SoC," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, sér. CF '23, Bologna, Italy : Association for Computing Machinery, août 2023, p. 389-390, ISBN : 9798400701405. DOI : 10.1145/3587135.3591436. adresse : https://doi.org/10.1145/3587135.3591436 (cité page 34).

[55] Yoann Bourgin. "L'allemand ARX Robotics lève 31 millions d'euros pour ses véhicules militaires autonomes." Consulté le 9 mai 2025. (2025), adresse : https://archive.ph/rzp8H (visité le 09/05/2025) (cité page 36).

[56] Naval Group. "Naval Group va réaliser un démonstrateur de drone sous-marin autonome pour la Direction Générale de l'Armement (DGA)." Consulté le 9 mai 2025. (jan. 2024), adresse : https://archive.ph/102AM (visité le 09/05/2025) (cité page 36).

[57] Les Échos. “Thales livre à la Marine le premier système autonome de lutte contre les mines.” Consulté le 9 mai 2025. (2025), adresse : https://archive.ph/h7V6g (visité le 09/05/2025) (cité page 36).

[58] Mohamed Amine Ferrag, Leandros Maglaras, Helge Janicke, Richard Smith, “Deep Learning Techniques for Cyber Security Intrusion Detection : A Detailed Analysis,” in *6th International Symposium for ICS & SCADA Cyber Security Research 2019*, sept. 2019. DOI : 10.14236/ewic/icscsr19.16 (cité page 36).

[59] Mohamed El Bouazzati, Russell Tessier, Philippe Tanguy, Guy Gogniat, “A Lightweight Intrusion Detection System against IoT Memory Corruption Attacks,” in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, Tallinn, Estonia : IEEE, 3 mai 2023, p. 118-123. DOI : 10.1109/DDECS57882.2023.10139718 (cité page 36).

[60] Nabila Farnaaz, M.A. Jabbar, “Random Forest Modeling for Network Intrusion Detection System,” *Procedia Computer Science*, t. 89, p. 213-217, 2016, ISSN : 1877-0509. DOI : https://doi.org/10.1016/j.procs.2016.06.047. adresse : https://www.sciencedirect.com/science/article/pii/S1877050916311127 (cité page 36).

[61] Gaurav Choudhary, Vishal Sharma, Ilsun You, Kangbin Yim, Ing-Ray Chen, Jin-Hee Cho, “Intrusion Detection Systems for Networked Unmanned Aerial Vehicles : A Survey,” in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, juin 2018, p. 560-565. DOI : 10.1109/IWCMC.2018.8450305 (cité page 37).

[62] Said Ouiazzane, Fatimazahra Barramou, Malika Addou, “Towards a Multi-Agent Based Network Intrusion Detection System for a Fleet of Drones,” *International Journal of Advanced Computer Science and Applications*, t. 11, n° 10, 2020. DOI : 10.14569/IJACSA.2020.0111044 (cité page 37).

[63] Nikola Kovacevic, Dorde Miseljic, Aleksa Stojkovic, “RISC-V vector processor for acceleration of machine learning algorithms,” in *2022 30th Telecommunications Forum (TELFOR)*, nov. 2022, p. 1-4. DOI : 10.1109/TELFOR56187.2022.9983779 (cité page 37).

[64] Ning Wu, Tao Jiang, Lei Zhang, Fang Zhou, Fen Ge, “A Reconfigurable Convolutional Neural Network-Accelerated Coprocessor Based on RISC-V Instruction Set,” *Electronics*, t. 9, n° 6, juin 2020, ISSN : 2079-9292. adresse : https://www.mdpi.com/2079-9292/9/6/1005 (cité page 37).

[65] Mohammadhossein Askarihemmat, Sean Wagner, Olexa Bilaniuk, Yassine Hariri, Yvon Savaria, Jean-Pierre David, "BARVINN : Arbitrary Precision DNN Accelerator Controlled by a RISC-V CPU," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, sér. ASPDAC '23, Tokyo, Japan : Association for Computing Machinery, jan. 2023, p. 483-489, ISBN : 9781450397834. DOI : 10.1145/3566097.3567872. adresse : https://doi.org/10.1145/3566097.3567872 (cité page 38).

[66] Alejandra Sanchez-Flores, Lluc Alvarez, Bartomeu Alorda-Ladaria, "A review of CNN accelerators for embedded systems based on RISC-V," in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, août 2022, p. 1-6. DOI : 10.1109/COINS54846.2022.9855006 (cité page 38).

[67] Sathwika Bavikadi, Purab Ranjan Sutradhar, Amlan Ganguly, Sai Manoj Pudukotai Dinakarrao, "Reconfigurable Processing-in-Memory Architecture for Data Intensive Applications," in *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, jan. 2024, p. 222-227. DOI : 10.1109/VLSID60093.2024.00043 (cité page 38).

[68] Bo Zhang, Shihui Yin, Minkyu Kim, "PIMCA : A Programmable In-Memory Computing Accelerator for Energy-Efficient DNN Inference," *IEEE Journal of Solid-State Circuits*, t. 58, nº 5, p. 1436-1449, mai 2023. DOI : 10.1109/JSSC.2022.3211290 (cité page 38).

[69] Yujeong Choi, Minsoo Rhu, "PREMA : A Predictive Multi-Task Scheduling Algorithm For Preemptible Neural Processing Units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, fév. 2020, p. 220-233. DOI : 10.1109/HPCA47549.2020.00027 (cité page 38).

[70] Jinwoo Choi, Yeonan Ha, Jounghoo Lee, "Enabling Fine-Grained Spatial Multitasking on Systolic-Array NPUs Using Dataflow Mirroring," *IEEE Transactions on Computers*, t. 72, nº 12, p. 3383-3398, déc. 2023. DOI : 10.1109/TC.2023.3299030 (cité page 38).

[71] Melisande Zonta-Roudes, Andres Meza, Nora Hinderling, "eXpect : On the Security Implications of Violations in AXI Implementations," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, sér. ICCAD '24, Newark Liberty International Airport Marriott, New York, NY, USA : Association for Computing Machinery, avr. 2025, ISBN : 9798400710773. DOI : 10.1145/3676536.3676844 (cité pages 39, 59).

[72] Melisande Zonta, Nora Hinderling, Shweta Shinde, "Xray : Detecting and Exploiting Vulnerabilities in Arm AXI Interconnects," in *2025 Design, Automation & Test in Europe Conference (DATE)*, avr. 2025, p. 1-7. DOI : 10.23919/DATE64628.2025.10992968 (cité page 39).

[73] Pasquale Davide Schiavone, Davide Rossi, Antonio Pullini, Alfio Di Mauro, Francesco Conti, Luca Benini, "Quentin : an Ultra-Low-Power PULPissimo SoC in 22nm FDX," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, oct. 2018, p. 1-3. DOI : 10.1109/S3S.2018.8640145 (cité page 40).

[74] Francesco Conti, Pasquale Davide Schiavone, Luca Benini, "XNOR Neural Engine : A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, juill. 2018, ISSN : 0278-0070. DOI : 10.1109/TCAD.2018.2857019 (cité page 40).

[75] Jiliang Zhang, Congcong Chen, Jinhua Cui, Keqin Li, "Timing Side-channel Attacks and Countermeasures in CPU Microarchitectures," *ACM Comput. Surv.*, t. 56, nº 7, avr. 2024, ISSN : 0360-0300. DOI : 10.1145/3645109. adresse : https://doi.org/10.1145/3645109 (cité page 43).

[76] Maria Mushtaq, Muhammad Asim Mukhtar, Vianney Lapotre, Muhammad Khurram Bhatti, Guy Gogniat, "Winter is here ! A decade of cache-based side-channel attacks, detection & mitigation for RSA," *Information Systems*, t. 92, p. 101 524, sept. 2020, ISSN : 0306-4379. DOI : https://doi.org/10.1016/j.is.2020.101524 (cité page 43).

[77] Yuval Yarom, Katrina Falkner, "FLUSH+RELOAD : A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA : USENIX Association, août 2014, p. 719-732. adresse : https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom (cité page 43).

[78] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, Ruby B. Lee, "Last-Level Cache Side-Channel Attacks are Practical," in *2015 IEEE Symposium on Security and Privacy*, mai 2015, p. 605-622. DOI : 10.1109/SP.2015.43 (cité page 43).

[79] Colin Percival, "Cache missing for fun and profit," 2005 (cité page 43).

[80] Dag Arne Osvik, Adi Shamir, Eran Tromer, "Cache Attacks and Countermeasures : The Case of AES," in *Topics in Cryptology – CT-RSA 2006*, D. POINTCHEVAL, éd., Berlin, Heidelberg : Springer Berlin Heidelberg, fév. 2006, p. 1-20, ISBN : 978-3-540-32648-9 (cité pages 43, 55).

[81] Eran Tromer, Dag Arne Osvik, Adi Shamir, "Efficient Cache Attacks on AES, and Countermeasures," *Journal of Cryptology*, t. 23, nº 1, p. 37-71, 2010. DOI : 10.1007/s00145-009-9049-y. adresse : https://link.springer.com/article/10.1007/s00145-009-9049-y (cité page 43).

[82] Antoon Purnal, Lukas Giner, Daniel Gruss, Ingrid Verbauwhede, "Systematic Analysis of Randomization-based Protected Cache Architectures," in *2021 IEEE Symposium on Security and Privacy (SP)*, mai 2021, p. 987-1002. DOI : 10.1109/SP40001.2021.00011 (cité page 43).

[83] Meng Wu, Shengjian Guo, Patrick Schaumont, Chao Wang, "Eliminating timing side-channel leaks using program repair," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, sér. ISSTA 2018, Amsterdam, Netherlands : Association for Computing Machinery, juill. 2018, p. 15-26, ISBN : 9781450356992. DOI : 10.1145/3213846.3213851. adresse : https://doi.org/10.1145/3213846.3213851 (cité page 44).

[84] Zhenghong Wang, Ruby B. Lee, "New cache designs for thwarting software cache-based side channel attacks," *SIGARCH Comput. Archit. News*, t. 35, nº 2, p. 494-505, juin 2007, ISSN : 0163-5964. DOI : 10.1145/1273440.1250723. adresse : https://doi.org/10.1145/1273440.1250723 (cité pages 44, 45, 49, 50, 55).

[85] Moinuddin K. Qureshi, "CEASER : Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, oct. 2018, p. 775-787. DOI : 10.1109/MICRO.2018.00068 (cité pages 44, 49, 50).

[86] Moinuddin K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, sér. ISCA '19, Phoenix, Arizona : Association for Computing Machinery, juin 2019, p. 360-371, ISBN : 9781450366694. DOI : 10.1145/3307650.3322246. adresse : https://doi.org/10.1145/3307650.3322246 (cité page 44).

[87] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, Stefan Mangard, "ScatterCache : Thwarting Cache Attacks via Cache Set Randomization," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA : USENIX Association, août 2019, p. 675-692. adresse : https://www.usenix.org/conference/usenixsecurity19/presentation/werner (cité page 44).

[88] Nils Wistoff, Gernot Heiser, Luca Benini, *fence.t.s : Closing Timing Channels in High-Performance Out-of-Order Cores through ISA-Supported Temporal Partitioning*, sept. 2024. arXiv : 2409.07576 [cs.CR]. adresse : https://arxiv.org/abs/2409.07576 (cité page 44).

[89] Mathieu Escouteloup, Ronan Lashermes, Jacques Fournier, Jean-Louis Lanet, "Under the Dome : Preventing Hardware Timing Information Leakage," in *Smart Card Research and Advanced Applications*, V. Grosso et T. Pöppelmann, éd., Cham : Springer International Publishing, mars 2022, p. 233-253, ISBN : 978-3-030-97348-3 (cité page 44).

[90] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, Dmitry Ponomarev, "Non-monopolizable caches : Low-complexity mitigation of cache side channel attacks," *ACM Trans. Archit. Code Optim.*, t. 8, nº 4, jan. 2012, ISSN : 1544-3566. DOI : 10.1145/2086696.2086714. adresse : https://doi.org/10.1145/2086696.2086714 (cité pages 44, 49, 50).

[91] Jean-Loup Hatchikian-Houdot, Pierre Wilke, Frédéric Besson, Guillaume Hiet, "Formal Hardware/Software Models for Cache Locking Enabling Fast and Secure Code," in *Computer Security – ESORICS 2024*, J. Garcia-Alfaro, R. Kozik, M. Choraś et S. Katsikas, éd., Cham : Springer Nature Switzerland, sept. 2024, p. 153-173, ISBN : 978-3-031-70896-1 (cité page 44).

[92] Iulia Bastys, Pauline Bolignano, Franco Raimondi, Daniel Schoepe, "Automatic Annotation of Confidential Data in Java Code," in *Foundations and Practice of Security*, E. Aïmeur, M. Laurent, R. Yaich, B. Dupont et J. Garcia-Alfaro, éd., Cham : Springer International Publishing, juin 2022, p. 146-161, ISBN : 978-3-031-08147-7 (cité page 47).

[93] Edward J. Schwartz, Thanassis Avgerinos, David Brumley, "All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask)," in *2010 IEEE Symposium on Security and Privacy*, mai 2010, p. 317-331. DOI : 10.1109/SP.2010.26 (cité page 47).

[94] Hans Winderix, Jan Tobias Mühlberg, Frank Piessens, "Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, sept. 2021, p. 667-682. DOI : 10.1109/EuroSP51992.2021.00050 (cité page 49).

[95] Fangfei Liu, Hao Wu, Kenneth Mai, Ruby B. Lee, "Newcache : Secure Cache Architecture Thwarting Cache Side-Channel Attacks," *IEEE Micro*, t. 36, nº 5, p. 8-16, oct. 2016. DOI : 10.1109/MM.2016.85 (cité pages 49, 50).

[96] Yao Wang, Andrew Ferraiuolo, Danfeng Zhang, Andrew C. Myers, G. Edward Suh, "SecDCP : secure dynamic cache partitioning for efficient timing channel protection," in *Proceedings of the 53rd Annual Design Automation Conference*, sér. DAC '16, Austin, Texas : Association for Computing Machinery, juin 2016, ISBN : 9781450342360. DOI : 10.1145/2897937.2898086. adresse : https://doi.org/10.1145/2897937.2898086 (cité pages 49, 50).

[97] Federico Canale, Tim Güneysu, Gregor Leander, Jan Philipp Thoma, Yosuke Todo, Rei Ueno, "SCARF : a low-latency block cipher for secure cache-randomization," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, sér. SEC '23, Anaheim, CA, USA : USENIX Association, août 2023, ISBN : 978-1-939133-37-3 (cité page 50).

[98] Moritz Peters, Nicolas Gaudin, Jan Philipp Thoma, "On The Effect of Replacement Policies on The Security of Randomized Cache Architectures," in *19th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2024)*, juill. 2024 (cité pages 50, 68).

[99] Paul Kocher, Jann Horn, Anders Fogh, "Spectre Attacks : Exploiting Speculative Execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, mai 2019, p. 1-19. DOI : 10.1109/SP.2019.00002 (cité page 52).

[100] Moritz Lipp, Michael Schwarz, Daniel Gruss, "Meltdown : reading kernel memory from user space," *Commun. ACM*, t. 63, nº 6, p. 46-56, mai 2020, ISSN : 0001-0782. DOI : 10.1145/3357033. adresse : https://doi.org/10.1145/3357033 (cité page 52).

[101] Muhammad Awais, Maria Mushtaq, Lirida Naviner, Florent Bruguier, Jawad Haj Yahya, Pascal Benoit, "Decoding Attack Behaviors by Analyzing Patterns in Instruction-Based Attacks using gem5," in *2024 International Workshop on Rapid System Prototyping (RSP)*, oct. 2024, p. 1-6. DOI : 10.1109/RSP64122.2024.10871078 (cité page 53).

[102] Wei Zheng, Ying Wu, Xiaoxue Wu, "A survey of Intel SGX and its applications," *Frontiers of Computer Science*, t. 15, nº 3, déc. 2020, ISSN : 2095-2236. DOI : 10.1007/s11704-019-9096-y. adresse : http://dx.doi.org/10.1007/s11704-019-9096-y (cité page 53).

[103] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, "Intel TDX Demystified : A Top-Down Approach," *ACM Comput. Surv.*, t. 56, nº 9, avr. 2024, ISSN : 0360-0300. DOI : 10.1145/3652597. adresse : https://doi.org/10.1145/3652597 (cité page 53).

[104] Sandro Pinto, Nuno Santos, "Demystifying Arm TrustZone : A Comprehensive Survey," *ACM Comput. Surv.*, t. 51, nº 6, jan. 2019, ISSN : 0360-0300. DOI : 10.1145/3291047. adresse : https://doi.org/10.1145/3291047 (cité page 53).

[105] Erhu Feng, Xu Lu, Dong Du, "Scalable Memory Protection in the PENGLAI Enclave," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, USENIX Association, juill. 2021, p. 275-294, ISBN : 978-1-939133-22-9. adresse : https://www.usenix.org/conference/osdi21/presentation/feng (cité pages 53, 54).

[106] Cesare Garlati, Sandro Pinto, "A Clean Slate Approach to Linux Security : RISC-V Enclaves," in *Embedded World Conference*, Nuremberg, Germany : Embedded World, fév. 2020. adresse : https://sandro2pinto.github.io/files/ew2020-linuxencl-riscv.pdf (cité page 53).

[107] Victor Costan, Ilia Lebedev, Srinivas Devadas, "Sanctum : Minimal Hardware Extensions for Strong Software Isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX : USENIX Association, août 2016, p. 857-874, ISBN : 978-1-931971-32-4. adresse : https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan (cité page 54).

[108] Antonio Munoz, Ruben Rios, Rodrigo Roman, Javier Lopez, "A survey on the (in)security of trusted execution environments," *Computers & Security*, t. 129, p. 103 180, juin 2023, ISSN : 0167-4048. DOI : https://doi.org/10.1016/j.cose.2023.103180. adresse : https://www.sciencedirect.com/science/article/pii/S0167404823000901 (cité page 54).

[109] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, Ahmad-Reza Sadeghi, "TIMBER-V : Tag-isolated memory bringing fine-grained enclaves to RISC-V.," in *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS'19)*, The Internet Society, fév. 2019. DOI : 10.14722/ndss.2019.23068 (cité page 54).

[110] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, "CURE : A Security Architecture with CUstomizable and Resilient Enclaves," in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, août 2021, p. 1073-1090, ISBN : 978-1-939133-24-3. adresse : https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani (cité page 54).

[111] Ernie Brickell, "Technologies to Improve Platform Security," in *CHES - Invited Talk*, Intel Corporation, Nara, Japan, sept. 2011 (cité page 55).

[112] Weiyi Wu, Bryan Ford, "Deterministically Deterring Timing Attacks in Deterland," mai 2016. arXiv : 1504.07070 [cs.OS]. adresse : https://arxiv.org/abs/1504.07070 (cité page 55).

[113] Amittai Aviram, Shu-Chun Weng, Sen Hu, Bryan Ford, "Efficient system-enforced deterministic parallelism," *Commun. ACM*, t. 55, nº 5, p. 111-119, mai 2012, ISSN : 0001-0782. DOI : 10.1145/2160718.2160742. adresse : https://doi.org/10.1145/2160718.2160742 (cité page 55).

[114] Yinqian Zhang, Michael K. Reiter, "Düppel : retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *CCS*, sér. CCS '13, Berlin, Germany : Association for Computing Machinery, nov. 2013, p. 827-838, ISBN : 9781450324779. DOI : 10.1145/2508859.2516741. adresse : https://doi.org/10.1145/2508859.2516741 (cité page 55).

[115] Mohit Tiwari, Xun Li, Hassan M G Wassel, Frederic T Chong, Timothy Sherwood, "Execution leases : A hardware-supported mechanism for enforcing strong non-interference," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, déc. 2009, p. 493-504. DOI : 10.1145/1669112.1669174 (cité page 55).

[116] Taesoo Kim, Marcus Peinado, Gloria Mainar-Ruiz, "STEALTHMEM : System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud," in *USENIX Security*, Bellevue, WA : USENIX Association, août 2012, p. 189-204, ISBN : 978-931971-95-9. adresse : https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/kim (cité page 55).

[117] Ziqiao Zhou, Michael K. Reiter, Yinqian Zhang, "A Software Approach to Defeating Side Channels in Last-Level Caches," in *CCS*, sér. CCS '16, Vienna, Austria : Association for Computing Machinery, oct. 2016, p. 871-882, ISBN : 9781450341394. DOI : 10.1145/2976749.2978324. adresse : https://doi.org/10.1145/2976749.2978324 (cité page 55).

[118] Ghada Dessouky, Tommaso Frassetto, Ahmad-Reza Sadeghi, "HybCache : Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments," in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, août 2020, p. 451-468, ISBN : 978-1-939133-17-5. adresse : https://www.usenix.org/conference/usenixsecurity20/presentation/dessouky (cité page 56).

[119] Daniel Townley, Kerem Arıkan, Yu David Liu, Dmitry Ponomarev, Oğuz Ergin, "Composable Cachelets : Protecting Enclaves from Cache Side-Channel Attacks," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA : USENIX Association, août 2022, p. 2839-2856, ISBN : 978-1-939133-31-1. adresse : https://www.usenix.org/conference/usenixsecurity22/presentation/townley (cité page 56).

[120] Lianying Zhao, He Shuang, Shengjie Xu, *SoK : Hardware Security Support for Trustworthy Execution*, 2019. arXiv : 1910.04957 [cs.CR]. adresse : https://arxiv.org/abs/1910.04957 (cité page 56).

[121] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, Ronald Perez, *SoK : Hardware-supported Trusted Execution Environments*, 2022. arXiv : 2205.12742 [cs.CR]. adresse : https://arxiv.org/abs/2205.12742 (cité page 56).

[122] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, Yinqian Zhang, "A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography," *ACM Comput. Surv.*, t. 54, nº 6, juill. 2021, ISSN : 0360-0300. DOI : 10.1145/3456629. adresse : https://doi.org/10.1145/3456629 (cité page 59).

[123] Xiaohan Zhang, Jinwen Wang, Yueqiang Cheng, "Interface-Based Side Channel in TEE-Assisted Networked Services," *IEEE/ACM Transactions on Networking*, t. 32, nº 1, p. 613-626, fév. 2024. DOI : 10.1109/TNET.2023.3294019 (cité page 59).

[124] Aruna Jayasena, Richard Bachmann, Prabhat Mishra, "EvilCS : An Evaluation of Information Leakage through Context Switching on Security Enclaves," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, mars 2024, p. 1-6. DOI : 10.23919/DATE58400.2024.10546809 (cité page 59).

[125] Cristiano Rodrigues, Daniel Oliveira, Sandro Pinto, "BUSted !!! Microarchitectural Side-Channel Attacks on the MCU Bus Interconnect," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, p. 3679-3696. DOI : 10.1109/SP54263.2024.00062 (cité page 59).

[126] Zhao Zhang, Yong Zhang, Da Guo, Lei Yao, Zhao Li, "SecFedNIDS : Robust defense for poisoning attack against federated learning-based network intrusion detection system," *Future Generation Computer Systems*, t. 134, p. 154-169, sept. 2022, ISSN : 0167-739X. DOI : https://doi.org/10.1016/j.future.2022.04.010. adresse : https://www.sciencedirect.com/science/article/pii/S0167739X22001339 (cité page 59).

[127] Yuan-Cheng Lai, Jheng-Yan Lin, Ying-Dar Lin, "Two-phase Defense Against Poisoning Attacks on Federated Learning-based Intrusion Detection," *Computers & Security*, t. 129, p. 103 205, juin 2023, ISSN : 0167-4048. DOI : https://doi.org/10.1016/j.cose.2023.103205. adresse : https://www.sciencedirect.com/science/article/pii/S0167404823001153 (cité page 59).

[128] Oussama Elmnaouri, **Pascal Cotret**, Vianney Lapotre, Loïc Lagadec, "Enhancing Keystone Security Against Cache Timing Attacks : A Modular Approach," in *Colloque 2025 du GDR SoC2*, Lorient, France, juin 2025. adresse : https://hal.science/hal-05056900 (cité page 67).

[129] Oussama Elmnaouri, **Pascal Cotret**, Vianney Lapotre, Loïc Lagadec, "Enforcing RISC-V TEE Security Against Cache Timing Attacks," in *International Workshops on Cryptographic architectures embedded in logic devices*, juin 2025 (cité page 67).

[130] Pierre Garreau, **Pascal Cotret**, Julien Francq, Jean-Christophe Cexus, Loïc Lagadec, "A survey on versatile embedded Machine Learning hardware acceleration," t. 167, oct. 2025, p. 103 501. DOI : https://doi.org/10.1016/j.sysarc.2025.103501. adresse : https://www.sciencedirect.com/science/article/pii/S1383762125001730 (cité page 68).

[131] Pierre Garreau, **Pascal Cotret**, Julien Francq, Jean-Christophe Cexus, Loïc Lagadec, "RISC-V Embedded AI for IDS Applications," in *RESSI 2024 : Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*, mai 2024 (cité page 68).

[132] Aya Jendoubi, Jean-Christophe Prévotet, Philippe Tanguy, **Pascal Cotret**, "Security of Dynamically Reconfigurable RISC-V Systems : I/O Attack Focus," in *39th Annual IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2025) : 32nd Reconfigurable Architecture Workshop*, Milan, Italy, juin 2025. adresse : https://hal.science/hal-05117047 (cité page 68).

[133] Aya Jendoubi, Jean-Christophe Prévotet, Philippe Tanguy, **Pascal Cotret**, "Enhancing Security in Heterogeneous Virtualized Systems : A Focus on I/O Attacks in the existence of IOMMU in a RISC-V architecture," in *GDR SoC2*, juin 2024 (cité page 68).

[134] Nicolas Gaudin, Vianney Lapôtre, **Pascal Cotret**, Guy Gogniat, "Verrouillage des lignes de cache pour la lutte contre les attaques par canaux auxiliaires exploitant les mémoires caches," in *Cyber On Board*, mars 2024 (cité page 68).

[135] Nicolas Gaudin, Vianney Lapôtre, **Pascal Cotret**, Guy Gogniat, "Cache locking against cache-based side-channel attacks," in *Ecole d'hiver Francophone sur les Technologies de Conception des Systèmes Embarqués Hétérogènes (FETCH)*, fév. 2024 (cité page 68).

[136] Nicolas Gaudin, Vianney Lapôtre, **Pascal Cotret**, Guy Gogniat, "A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, juill. 2024 (cité page 68).

[137] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "Securing a high-level language virtual machine through its ISA : Pharo as a case study," in *GDR SoC2*, juin 2021 (cité page 68).

[138] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, Rob Stewart, "Benchmarking Quantized Neural Networks on FPGAs with FINN," in *SLOHA - DATE Friday Workshop on System-level Design Methods for Deep Learning on Heterogeneous Architectures*, fév. 2021 (cité page 68).

[139] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "JIT Compiler Security through Low-Cost RISC-V Extension," in *RAW - 30th Reconfigurable Architectures Workshop*, mai 2023 (cité page 68).

[140] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "War on JITs : Software-Based Attacks and Hybrid Defenses for JIT Compilers - A Comprehensive Survey," *ACM Comput. Surv.*, avr. 2025, ISSN : 0360-0300. DOI : 10.1145/3731598. adresse : https://doi.org/10.1145/3731598 (cité page 68).

[141] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "Monitoring information flows in heterogeneous SoCs with a dedicated coprocessor," in *GDR SoC-SiP*, juin 2018 (cité page 68).

[142] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "A framework for efficient DIFT in real-world SoCs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL) - Demo session*, juill. 2017, p. 1-2 (cité page 68).

[143] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, "A MIPS-based coprocessor for information flow tracking in ARM SoCs," in *2018 International Conference on Reconfigurable Computing and FPGAs (Reconfig)*, déc. 2018, p. 1-8 (cité page 68).

[144] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, "A novel lightweight hardware-assisted static instrumentation approach for ARM SoC using debug components," in *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, déc. 2018, p. 1-6 (cité page 68).

[145] Muhammad Abdul Wahab, **Pascal Cotret**, "Pwning ARM Debug Components for Sec-Related Stuff," in *Hack In the Box Security Conference - CommSec track*, avr. 2017 (cité page 68).

[146] **Pascal Cotret**, Muhammad Abdul Wahab, "TrustZone is not enough - Hijacking debug components for embedded security," in *Chaos Communication Congress*, déc. 2017 (cité page 68).

[147] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "ARMHEx : a hardware extension for information flow tracking on ARM-based platforms," in *RESSI 2017 (Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des systèmes d'information)*, mai 2017 (cité page 68).

[148] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "A portable approach for SoC-based Dynamic Information Flow Tracking implementations," in *GDR SoC-SiP*, juin 2016 (cité page 68).

[149] Muhammad A. Wahab, **Pascal Cotret**, Mounir N. Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "ARMHEx : a hardware extension for information flow tracking on ARM-based platforms," in *GDR SoC-SiP*, juin 2017 (cité page 68).

[150] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "Towards a hardware-assisted information flow tracking ecosystem for ARM processors," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, août 2016, p. 1-2. DOI : 10.1109/FPL.2016.7577396 (cité page 68).

[151] Muhammad Abdul Wahab, **Pascal Cotret**, "A hardware coprocessor for Zynq-based Dynamic Information Flow Tracking," in *Cryptographic Architectures Embedded in Reconfigurable Devices, International Workshops on*, juin 2016 (cité page 68).

[152] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "ARMHEx : A hardware extension for DIFT on ARM-based SoCs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, juill. 2017, p. 1-7 (cité page 68).

Publications

Article en cours d'évaluation dans une revue

[39] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "JITDomain : Instruction-Level JIT Code Isolation," *Microprocessors and Microsystems*, Elsevier, éd., août 2025, article en révision mineure (cité pages 26, 68).

Articles dans des revues internationales

[130] Pierre Garreau, **Pascal Cotret**, Julien Francq, Jean-Christophe Cexus, Loïc Lagadec, "A survey on versatile embedded Machine Learning hardware acceleration," t. 167, oct. 2025, p. 103 501. DOI : https://doi.org/10.1016/j.sysarc.2025.103501. adresse : https://www.sciencedirect.com/science/article/pii/S1383762125001730 (cité page 68).

[140] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "War on JITs : Software-Based Attacks and Hybrid Defenses for JIT Compilers - A Comprehensive Survey," *ACM Comput. Surv.*, avr. 2025, ISSN : 0360-0300. DOI : 10.1145/3731598. adresse : https://doi.org/10.1145/3731598 (cité page 68).

[153] **Pascal Cotret**, Guy Gogniat, Martha Johanna Sepúlveda Flórez, "Protection of heterogeneous architectures on FPGAs : An approach based on hardware firewalls," *Microprocessors and Microsystems*, t. 42, p. 127-141, mai 2016, ISSN : 0141-9331. DOI : http://dx.doi.org/10.1016/j.micpro.2016.01.013. adresse : http://www.sciencedirect.com/science/article/pii/S0141933116000259.

Chapitres d'ouvrages

[154] Eduardo Wanderley, Romain Vaslin, Jérémie Crenne, "Security FPGA Analysis," English, in *Security Trends for FPGAS*, B. Badrignans, J. L. Danger, V. Fischer, G. Gogniat et L. Torres, éd., Springer Netherlands, 2011, p. 7-46, ISBN : 978-94-007-1337-6. DOI : 10.1007/978-94-007-1338-3_2. adresse : http://dx.doi.org/10.1007/978-94-007-1338-3_2.

Articles dans des revues nationales

[155] **Pascal Cotret**, Guy Gogniat, "Protection des architectures hétérogènes sur FPGA : une approche par pare-feux matériels," *Techniques de l'Ingenieur*, Référence IN175 -10 pages, fév. 2014. adresse : https://hal.inria.fr/hal-00866646.

Conférences/communications invitées

[156] **Pascal Cotret**, "Monitoring program execution (and more) on ARM processors," in *Toulouse Hacking Convention*, mars 2018, p. 1-2.

[157] **Pascal Cotret**, "Towards a hardware-assisted information flow Tracking Approach for ARM Processors," in *France/Japan Cybersecurity workshop*, mars 2016, p. 1-2.

Articles dans des conférences/workshops à audience internationale avec actes et comité de sélection

[33] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "Gigue : A JIT Code Binary Generator for Hardware Testing," in *2023 Workshop on Virtual Machines and Language Implementations*, oct. 2023 (cité pages 18, 68).

[98] Moritz Peters, Nicolas Gaudin, Jan Philipp Thoma, "On The Effect of Replacement Policies on The Security of Randomized Cache Architectures," in *19th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS 2024)*, juill. 2024 (cité pages 50, 68).

[132] Aya Jendoubi, Jean-Christophe Prévotet, Philippe Tanguy, **Pascal Cotret**, "Security of Dynamically Reconfigurable RISC-V Systems : I/O Attack Focus," in *39th Annual IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2025) : 32nd Reconfigurable Architecture Workshop*, Milan, Italy, juin 2025. adresse : https://hal.science/hal-05117047 (cité page 68).

[136] Nicolas Gaudin, Vianney Lapôtre, **Pascal Cotret**, Guy Gogniat, "A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, juill. 2024 (cité page 68).

[138] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, Rob Stewart, "Benchmarking Quantized Neural Networks on FPGAs with FINN," in *SLOHA - DATE Friday Workshop on System-level Design Methods for Deep Learning on Heterogeneous Architectures*, fév. 2021 (cité page 68).

[139] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "JIT Compiler Security through Low-Cost RISC-V Extension," in *RAW - 30th Reconfigurable Architectures Workshop*, mai 2023 (cité page 68).

[142] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "A framework for efficient DIFT in real-world SoCs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL) - Demo session*, juill. 2017, p. 1-2 (cité page 68).

[143] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, "A MIPS-based coprocessor for information flow tracking in ARM SoCs," in *2018 International Conference on Reconfigurable Computing and FPGAs (Reconfig)*, déc. 2018, p. 1-8 (cité page 68).

[144] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, "A novel lightweight hardware-assisted static instrumentation approach for ARM SoC using debug components," in *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, déc. 2018, p. 1-6 (cité page 68).

[150] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "Towards a hardware-assisted information flow tracking ecosystem for ARM processors," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, août 2016, p. 1-2. DOI : 10.1109/FPL.2016.7577396 (cité page 68).

[152] Muhammad Abdul Wahab, **Pascal Cotret**, Mounit Nasr Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "ARMHEx : A hardware extension for DIFT on ARM-based SoCs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, juill. 2017, p. 1-7 (cité page 68).

[158] Jean-Loup Hatchikian-Houdot, Nicolas Gaudin, **Pascal Cotret**, "Work in Progress : Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections," in *SILM'23 - IEEE EuroSP workshop*, juill. 2023.

[159] Quentin Ducasse, Guille Polito, Pablo Tesone, **Pascal Cotret**, Loïc Lagadec, "Porting a JIT compiler to RISC-V : Challenges and Opportunities," in *MPLR - Managed Programming Languages and Runtimes 2022*, sept. 2022.

[160] **Pascal Cotret**, Stéphane Chevobbe, Mehdi Darouich, "Embedded wavelet-based face recognition under variable position," in *SPIE Electronic Imaging*, t. 9400, SPIE Electronic Imaging, fév. 2015, 94000A-94000A-12. DOI : 10.1117/12.2083046. adresse : http://dx.doi.org/10.1117/12.2083046.

[161] **Pascal Cotret**, Guy Gogniat, Jean-Philippe Diguet, Jérémie Crenne, "Lightweight reconfiguration security services for AXI-based MPSoCs," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, août 2012, p. 655-658. DOI : 10.1109/FPL.2012.6339233.

[162] **Pascal Cotret**, Florian Devic, Guy Gogniat, Benoit Badrignans, Benoit Torres, "Security enhancements for FPGA-based MPSoCs : A boot-to-runtime protection flow for an embedded Linux-based system," in *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, juill. 2012, p. 1-8. DOI : 10.1109/ReCoSoC.2012.6322896.

[163] **Pascal Cotret**, Jérémie Crenne, Guy Gogniat, Jean-Philippe Diguet, "Bus-based MPSoC Security through Communication Protection : A Latency-efficient Alternative," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, avr. 2012, p. 200-207. DOI : 10.1109/FCCM.2012.42.

[164] Jérémie Crenne, **Pascal Cotret**, Guy Gogniat, Russell Tessier, Jean-Philippe Diguet, "Efficient key-dependent message authentication in reconfigurable hardware," in *Field-Programmable Technology (FPT), 2011 International Conference on*, déc. 2011, p. 1-6. DOI : 10.1109/FPT.2011.6132722.

[165] **Pascal Cotret**, Jérémie Crenne, Guy Gogniat, Jean-Philippe Diguet, Lubos Gaspar, Guillaume Duc, "Distributed Security for Communications and Memories in a Multiprocessor Architecture," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, mai 2011, p. 326-329. DOI : 10.1109/IPDPS.2011.158.

[166] Lubos Gaspar, Viktor Fischer, Florent Bernard, Lilian Bossuet, **Pascal Cotret**, "HCrypt : A Novel Concept of Crypto-processor with Secured Key Management," in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, déc. 2010, p. 280-285. DOI : 10.1109/ReConFig.2010.38.

Articles dans des conférences/workshops à audience internationale avec comité de sélection

[167] **Pascal Cotret**, Vipin Kizheppatt, Christophe Moy, "Multi-standard OFDM transceiver for heterogeneous system-on-chips," in *WinnComm Europe*, oct. 2016.

Articles dans des conférences/workshops à audience nationale avec actes et comité de sélection

[168] Valérie Viet Viem Tong, Benoît Fournier, Guillaume Fournier, Leopold Ouairy, **Pascal Cotret**, Gilles Guette, "Dis, c'est quoi là haut dans le ciel ? - C'est un Linux, mon petit," in *Magazine MISC*, juill. 2019.

[169] Guillaume Fournier, Paul Audren de Kerdrel, **Pascal Cotret**, Valérie Viet Triem Tong, "DroneJack : kiss your drones goodbye !" In *SSTIC (Symposium sur la sécurité des technologies de l'information et des communications)*, juin 2017.

Articles dans des conférences/workshops à audience nationale avec comité de sélection

[170] Guillaume Fournier, Pierre Matoussowsky, **Pascal Cotret**, "Hit the KeyJack : stealing data from your daily wireless devices incognito," in *Journées C&ESAR 2016*, nov. 2016.

[171] **Pascal Cotret**, Stéphane Chevobbe, Mehdi Darouich, "Reconnaissance faciale basée sur les ondelettes robuste et optimisée pour les systèmes embarqués," in *Colloque GRETSI*, sept. 2015.

[172] **Pascal Cotret**, Jérémie Crenne, Guy Gogniat, "Sécurisation des communications dans une architecture multiprocesseur," in *MajecSTIC (MAnifestation des JEunes Chercheurs en Sciences et Technologies de l'Information et de la Communication)*, oct. 2010, p. 163-170.

Communications orales (sans acte) ou posters ou séminaires autres

[131] Pierre Garreau, **Pascal Cotret**, Julien Francq, Jean-Christophe Cexus, Loïc Lagadec, "RISC-V Embedded AI for IDS Applications," in *RESSI 2024 : Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*, mai 2024 (cité page 68).

[133] Aya Jendoubi, Jean-Christophe Prévotet, Philippe Tanguy, **Pascal Cotret**, "Enhancing Security in Heterogeneous Virtualized Systems : A Focus on I/O Attacks in the existence of IOMMU in a RISC-V architecture," in *GDR SoC2*, juin 2024 (cité page 68).

[134] Nicolas Gaudin, Vianney Lapôtre, **Pascal Cotret**, Guy Gogniat, "Verrouillage des lignes de cache pour la lutte contre les attaques par canaux auxiliaires exploitant les mémoires caches," in *Cyber On Board*, mars 2024 (cité page 68).

[135] Nicolas Gaudin, Vianney Lapôtre, **Pascal Cotret**, Guy Gogniat, "Cache locking against cache-based side-channel attacks," in *Ecole d'hiver Francophone sur les Technologies de Conception des Systèmes Embarqués Hétérogènes (FETCH)*, fév. 2024 (cité page 68).

[137] Quentin Ducasse, **Pascal Cotret**, Loïc Lagadec, "Securing a high-level language virtual machine through its ISA : Pharo as a case study," in *GDR SoC2*, juin 2021 (cité page 68).

[141] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "Monitoring information flows in heterogeneous SoCs with a dedicated coprocessor," in *GDR SoC-SiP*, juin 2018 (cité page 68).

[147] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "ARMHEx : a hardware extension for information flow tracking on ARM-based platforms," in *RESSI 2017 (Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des systèmes d'information)*, mai 2017 (cité page 68).

[148] Muhammad Abdul Wahab, **Pascal Cotret**, Mounir Nasr Allah Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "A portable approach for SoC-based Dynamic Information Flow Tracking implementations," in *GDR SoC-SiP*, juin 2016 (cité page 68).

[149] Muhammad A. Wahab, **Pascal Cotret**, Mounir N. Allah, Guillaume Hiet, Vianney Lapôtre, Guy Gogniat, "ARMHEx : a hardware extension for information flow tracking on ARM-based platforms," in *GDR SoC-SiP*, juin 2017 (cité page 68).

[151] Muhammad Abdul Wahab, **Pascal Cotret**, "A hardware coprocessor for Zynq-based Dynamic Information Flow Tracking," in *Cryptographic Architectures Embedded in Reconfigurable Devices, International Workshops on*, juin 2016 (cité page 68).

[173] Guillermo Polito, Stéphane Ducasse, Pablo Tesone, "Remarkable Challenges of High-Performance Language Virtual Machines," in *Research Report Inria Lille - Nord Europe*, sept. 2022.

[174] **Pascal Cotret**, Guillaume Hiet, Guy Gogniat, "HardBlare : an efficient hardware-assisted DIFC for non-modified embedded processors," in *CHES (Workshop on Cryptographic Hardware and Embedded Systems)*, juill. 2015.

[175] **Pascal Cotret**, Guy Gogniat, Jean-Philippe Diguet, "Self-configuration of latency-efficient security enhancements for MPSoC communications monitoring," in *GDR SoC-SiP*, juin 2012.

[176] **Pascal Cotret**, Guy Gogniat, Jean-Philippe Diguet, Jérémie Crenne, "Self-reconfigurable security-enhanced communications in FPGA-based MPSoCs," in *Cryptographic Architectures Embedded in Reconfigurable Devices, International Workshops on*, juin 2012.

[177] **Pascal Cotret**, Jérémie Crenne, Guy Gogniat, Jean-Philippe Diguet, "Protecting communications in bus-based MPSoCs using hardware firewalls," in *Cryptographic Architectures Embedded in Reconfigurable Devices, International Workshops on*, juin 2011.

[178] **Pascal Cotret**, Jérémie Crenne, Guy Gogniat, Jean-Philippe Diguet, "A case study for distributed and efficient protection of communications in reconfigurable embedded systems," in *GDR SoC-SiP*, juin 2011.

[179] **Pascal Cotret**, Jérémie Crenne, Guy Gogniat, "Secured communications within a multiprocessor architecture," in *GDR SoC-SiP*, juin 2010.