Contributions à la sécurité des systèmes embarqués face aux attaques logiques et physiques

Vianney Lapotre

COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université Bretagne Sud
ubs

HABILITATION À DIRIGER DES RECHERCHES

UNIVERSITÉ DE BRETAGNE SUD

ÉCOLE DOCTORALE N° 644
Mathématiques et Sciences et Technologies de l'Information et de la Communication en Bretagne Océane
Spécialité : *Sciences et technologies de l'information et de la communication (S.T.I.C.)*

Par

Vianney LAPÔTRE

Contributions à la sécurité des systèmes embarqués face aux attaques logiques et physiques

soutenue à Lorient , le 4 Juillet 2023
Unité de recherche : Lab-STICC

Rapporteurs avant soutenance :

	Lilian BOSSUET	Professeur - Université Jean Monnet
	Karine HEYDEMANN	Maître de Conférences - Sorbonne Université / Thales
	Sylvain GUILLEY	Professeur - Télécom Paris / Secure-IC

Composition du Jury :

Président :	Abdoulaye GAMATIÉ	Directeur de Recherche - CNRS
Examinateurs :	Lilian BOSSUET	Professeur - Université Jean Monnet
	Karine HEYDEMANN	Maître de Conférences - Sorbonne Université / Thales
	Sylvain GUILLEY	Professeur - Télécom Paris / Secure-IC
	Guy GOGNIAT	Professeur - Université de Bretagne-Sud

Avant propos

Ce manuscrit présente une synthèse de mes travaux de recherche, mes activités d'enseignement et administratives depuis ma nomination en tant que Maître de Conférences au sein de l'Université Bretagne Sud en septembre 2014. Depuis, je réalise mes enseignements au sein de l'École Nationale Supérieure d'Ingénieurs de Bretagne Sud. Entre 2014 et 2020, j'ai mené mes travaux au sein de l'équipe Méthodes et Outils pour les Circuits et Systèmes (MOCS) du Laboratoire Lab-STICC. Puis, suite à une restructuration du laboratoire en 2020, j'ai intégré l'équipe architectures matérielles et outils de CAO (ARCAD) dont je suis aujourd'hui le responsable.

Ce manuscrit présente également, mais de façon plus succincte, mes travaux de recherche entre 2010 et 2014, période durant laquelle j'ai réalisé mes travaux de thèse au sein du laboratoire Lab-STICC et mes travaux post-doctoraux au sein du laboratoire LIRMM ainsi que des enseignements, entre 2010 et 2013, au sein de l'UFR sciences et sciences de l'ingénieur de Lorient.

Depuis près de 9 ans maintenant, mes travaux s'intéressent au domaine de la conception de systèmes numériques de confiance. Parallèlement, je me suis impliqué dans la vie pédagogique de l'ENSIBS à travers le développement d'enseignements nouveaux en lien avec la sécurité des systèmes numériques, en participant au développement de la filière Cyberdéfense de l'école créée en 2013 et en prenant en charge la coordination du recrutement des nouveaux étudiants depuis 2015.

Ce document est organisé en cinq parties afin de couvrir l'ensemble de mes activités :

Première partie - Synthèse des travaux

Deuxième partie - Contributions à l'axe : implémentation de fonctions cryptographiques

Troisième Partie - Contributions à l'axe : collaboration logiciel / matériel pour la sécurité des systèmes embarqués

Quatrième Partie - Contributions à l'axe : sécurité des processeurs embarqués

Cinquième Partie - Conclusion et perspectives

Table des matières

Première partie

Synthèse des travaux

Cette première partie présente de façon complète l'ensemble de mon parcours en mettant en avant toutes ses contributions et originalités. Elle permet d'appréhender mes réalisations et de comprendre mes motivations.

Résumé du dossier

Curriculum Vitae	**Fonction actuelle** **Maître de Conférences à l'Université Bretagne Sud** Enseignement à l'École Nationale Supérieure d'Ingénieurs de Bretagne Sud (ENSIBS) Recherche au Laboratoire des Sciences et Technologies de l'Information, de la Communication et de la Connaissance (Lab-STICC), UMR CNRS 6285 **Diplômes** **Doctorat Sciences et Techniques de l'Information et de la Communication** Université de Bretagne Sud, Lorient / Laboratoire Lab-STICC Soutenue le 20 Novembre 2013 Mention *très honorable* et label *doctorat européen* **Master Sciences Technologies Santé, mention Mathématiques - STIC** Université de Bretagne Sud, Lorient, obtenu en 2010, mention très bien Spécialité Électronique et informatique Industrielle **Licence de Sciences et Technologies, mention Sciences de l'ingénieur** Université de Bretagne Sud, Lorient, obtenu en 2009, mention bien **Expériences professionnelles** Depuis Septembre 2014 - **Maître de Conférences à l'Université Bretagne Sud** Prime d'Encadrement Doctoral et de Recherche (PEDR 2018 - 2022) Novembre 2013 - Août 2014 - **Chercheur contractuel CNRS** Laboratoire LIRMM - CNRS, UMR 5506 - Montpellier Octobre 2010 – Septembre 2013 - **Doctorant (bourse CDE)** Université Bretagne Sud / Laboratoire Lab-STICC - CNRS, UMR 6285 - Lorient
Enseignement	Sécurité matérielle, microprocesseur et langage d'assemblage, Linux embarqué, sécurité de la microarchitecture, programmation en langage C, programmation défensive et anti-reverse, réseaux informatiques, électronique numérique et analogique
Recherche	Sécurité des processeurs embarqués, sécurité des systèmes embarqués, protections face à des attaques logiques et physiques, architectures reconfigurables
Laboratoire	Laboratoire Lab-STICC, UMR CNRS 6285, Université Bretagne Sud
Responsabilités administratives	Responsable du recrutement de l'école d'ingénieurs ENSIBS (depuis 2015) Référent pédagogique de la première année du cycle ingénieur en Cyberdéfense de l'ENSIBS (2015-2018) Co-représentant de l'ENSIBS au Pôle d'Excellence Cyber (PEC) de la région Bretagne (2014-2018) Élu au conseil de laboratoire du Lab-STICC (depuis avril 2022) Responsable de l'équipe de recherche ARCAD du laboratoire Lab-STICC (depuis septembre 2021) Animateur du groupe de travail RISC-V de l'équipe ARCAD du laboratoire Lab-STICC (2020 - 2021) Correspondant pour l'équipe MOCS au sein de l'axe transverse Cybersécurité du Lab-STICC (2015-2019) Élu au conseil scientifique de l'Université Bretagne Sud (2012-2013)

Responsabilités scientifiques	**Comité de sélection MCF** Université de Bretagne Occidentale (2018) **Direction/édition d'ouvrages** Journal Applied sciences, special issue on side channel attacks in embedded system, MDPI, 2021 **Comité de programme de conférences internationales** ICCS (2015 à 2019), ARC (2015 à 2018), CS2 (2015, 2016, 2019), LASCAS (2015 à 2022), NEWCAS (2015), MCSoC (2015 à 2022), WCSIoT (2015), BEC (2016, 2018), EAMTA (2016), PRIME (2017), FPL (2017 à 2022), SBCCI (2018, 2020 à 2022), SECUWARE (2020), APCCAS (2020), SILM (2020 à 2022), PROOFS (2022) **Comité de lecture de conférences internationales** SoC (2013), FCCM (2013), ESLsyn (2014), CARI (2014), ReCoSoC (2014), DATE (2015, 2016, 2018 à 2020), ICASSP (2016, 2017, 2019), GLSVLSI (2016), ASAP (2016), SiPS (2017 à 2020), DAC (2017, 2021, 2022), NEWCAS (2019), FDL (2019) **Comité de programme de conférence nationale** Compas (2022) **Comité de lecture de conférence nationale** Compas (2016) **Comité de lecture de journaux internationaux** Springer Journal of Supercomputing (2016, 2017, 2021), Elsevier MICPRO (2016, 2017 à 2022), IEEE Transactions on Parallel and Distributed Systems - TPDS (2016, 2017), IEEE Transactions on Computer-Aided Design - TCAD (2017, 2020, 2021), IEEE Transactions on Consumer Electronics - TCE (2022), IEEE Transactions on Computers - TC (2018, 2020, 2022), ACM Transactions on Embedded Computing Systems - TECS (2019), IEEE Transactions on Very Large Scale Integration Systems - TVLSI (2016, 2017, 2021, 2022), IEEE Embedded Systems Letters (2019, 2022), Integration the VLSI Journal, Elsevier (2020), IEEE ACCESS (2020), Elsevier Journal of Systems Architecture (2022), **Comité d'organisation d'événements internationaux** ReCoSoC 2014, FPL 2017, SiPS 2017, CryptArchi 2018, Alchemy 2019 **Comité d'organisation d'événements nationaux** Compas 2016, ARCHI 2019, journée "Sécurité, fiabilité et test des SoC2 : challenges et opportunités dans l'ère de l'Intelligence Artificielle", 2019 **Modération dans des conférences internationales** ReCoSoC 2014, FPL 2017 **Expertise scientifique** ANR (2018, 2019, 2021), Région Auvergne Rhône-Alpes (2020), AGP - campagne de bourses de thèse du pôle de recherche cyber (2022)
Encadrements	9 co-encadrements de thèses (6 soutenues et 3 en cours) 2 encadrements de post-doc 1 encadrement d'ingénieur 6 encadrements de Master
Contrats et Collaborations	**Contrats de recherche** **Securité RISC-V (2022)**, micro-projet LATERAL, Lab-STICC, Thales **RISC-V (2022)**, micro-projet GIS Cormorant, Lab-STICC, Thales **SCRATCHS (2021-2024)**, LabEX CominLabs, Lab-STICC, IRISA **Usine du futur (2021-2027)**, CPER, Lab-STICC **INS3PECT (2017)**, CNRS PEPS, Lab-STICC, I3S, LEAT, LIG **HomCrypt (2016-2017)**, CNRS PEPS SISC INS2I, Lab-STICC **CyberSSI (2015-2020)**, CPER, Lab-STICC **HardBlare (2015-2019)**, LabEX CominLabs, Lab-STICC, IRISA **PPI Cyberdéfense (2015-2017)**, UBS, Lab-STICC, IRISA, CRPCC, IREA, LMBA **TSUNAMY (2013-2017)**, ANR, Lab-STICC, CEA LIST, LIP6, LabHC **Collaborations académiques** **SECUREORDO (2015)**, bourse UBS, ITU (Pakistan) **e-health.SECURE (2017-2019)**, PHC PERIDOT, ITU (Pakistan) **PROTECTCACHE (2017-2018)**, bourses d'excellence Eiffel, ITU (Pakistan) **SAFEPARA (2015)**, bourse UBS, RUB (Allemagne) **MULTISEC (2015)**, Bourse UBS et RUB, RUB (Allemagne) **CRYPHW (2016)**, bourse UBS et UMASS, UMASS (USA)
Publications	12 revues scientifiques internationales 1 revue scientifique nationale 3 participations à des ouvrages scientifiques internationaux 27 publications en conférences internationales 5 publications en conférences nationales

1 Curriculum Vitae

1.1 État civil

LAPÔTRE Vianney
Né le 07 Juillet 1987 à Calais (62)
Nationalité française
35 ans, célibataire

adresse professionnelle
Laboratoire Lab-STICC - CNRS, UMR 6285
Centre de Recherche Christiaan Huygens
Rue de Saint-Maudé
CS 7030 - 56321 Lorient CEDEX - FRANCE
email : vianney.lapotre@univ-ubs.fr
tel : 02 97 87 45 68
web : `http://labsticc.univ-ubs.fr/~lapotre/`

1.2 Grades et titres universitaires

2013
Doctorat Sciences et Techniques de l'Information et de la Communication
Université de Bretagne Sud, Lorient
Laboratoire Lab-STICC (Laboratoire des Sciences et Techniques de l'Information de la Communication et de la Connaissance)
Titre : Toward dynamically reconfigurable high throughput multiprocessor Turbo decoder in a multimode and multi-standard context.
Soutenue le 20 Novembre 2013
Mobilité : du 01 Juin 2012 au 30 Novembre 2012, Université de Bochum (Allemagne)
Mention : très honorable
Label : doctorat européen

2010
Master Sciences Technologies Santé, mention Mathématiques - STIC
Spécialité : Électronique et Informatique Industrielle
Mention : très bien - Université de Bretagne Sud, Lorient

2009
Licence de Sciences et Technologies, mention Sciences de l'Ingénieur
Mention : bien - Université de Bretagne Sud, Lorient

2007
Diplôme Universitaire de Technologie
Spécialité : Génie Électronique et Informatique Industrielle - Mention : bien
institut Universitaire de Technologie de Calais-Boulogne
Université du Littoral - Côte d'Opale, Calais

2005
Diplôme du Baccalauréat Technologique
Série : Sciences et Technologies Industrielles
Spécialité : Génie Électronique - Mention : très bien

1.3 Situations successives

Septembre 2014 – Aujourd'hui
Maître de Conférences (PEDR depuis octobre 2018)
Section CNU 61 (qualifié CNU 61 en mars 2014)
Université Bretagne Sud
École d'ingénieurs ENSIBS - Spécialité Cyberdéfense
Laboratoire Lab-STICC - CNRS, UMR 6285 - Lorient

Novembre 2013 – Août 2014
Chercheur contractuel
Centre National de la Recherche Scientifique (CNRS)
Laboratoire LIRMM - CNRS, UMR 5506 - Montpellier

Octobre 2010 – Septembre 2013
Doctorant avec mission d'enseignement - bourse CDE
Université Bretagne Sud
Laboratoire Lab-STICC - CNRS, UMR 6285 - Lorient

1.4 Résumé

J'ai obtenu mon doctorat en Sciences et Techniques de l'Information et de la Communication en 2013 à l'Université Bretagne Sud, au sein du laboratoire des Sciences et Techniques de l'Information de la Communication et de la Connaissance (Lab-STICC). De 2010 à 2013, j'ai réalisé des missions d'enseignement au sein de l'UFR Sciences et Sciences de l'Ingénieur de l'Université Bretagne Sud. De plus, j'ai réalisé une mobilité de 6 mois (Juin à Novembre 2012) au sein du laboratoire ESIT de l'Université de Bochum (Allemagne).

Depuis 2014, je suis Maître de Conférences au sein de l'École Nationale Supérieure d'Ingénieurs de Bretagne Sud (ENSIBS) de l'Université Bretagne Sud. J'effectue principalement des enseignements au sein de la spécialité Cyberdéfense en informatique industrielle (système d'exploitation, programmation) et électronique numérique (sécurité matérielle, architecture des processeurs).

Je réalise mes activités de recherche dans l'équipe *architectures matérielles et outils de CAO* (ARCAD) du laboratoire Lab-STICC. Mes travaux concernent l'étude et la conception d'architectures matérielles sécurisées pour les systèmes embarqués.

J'ai publié une trentaine d'articles dans des conférences majeures du domaine des systèmes embarqués et de l'architecture des systèmes numériques (FPL, FPT, ASP-DAC, ISCAS...) ainsi que dans différents ouvrages et revues scientifiques (IEEE TVLSI, IEEE TC, ACM TECS, EURASIP, JISA...). Actuellement et depuis plusieurs années, je participe à de nombreux comités de programmes de conférences internationales (FPL, SBCCI, MC-SoC, LASCAS...). Je suis également relecteur pour différentes revues internationales (IEEE TVLSI, IEEE TC, MICPRO...). J'ai été éditeur invité d'un numéro spécial du journal MDPI *Aplied Science Special Issue on Side Channel Attacks in Embedded Systems*. J'ai également participé en tant qu'expert à différents appels à projets nationaux (ANR, Auvergne-Rhône-Alpes, région Bretagne).

Je participe à plusieurs projets de recherche (ANR, LabEx Cominlabs). Je participe aussi à l'activité nationale dans les domaines de la sécurité et de la conception de systèmes numériques embarqués (GDR SOC2, GDR Sécurité Informatique).

1.5 Encadrements doctoraux et post-doctoraux

La liste ci-dessous résume les 9 co-encadrements de thèses effectués depuis 2014 (dont 3 en cours) ainsi que la situation professionnelle actuelle de ces étudiants. À cette liste s'ajoute le travail avec deux Post-Doctorants, un ingénieur et l'encadrement de 7 étudiants de Master.

1.5.1 Thèses en cours

Hongwei Zhao
Années de thèse : depuis 2022
Titre : Architecture de communication sécurisée d'un SoC vis-à-vis des attaques physiques et logiques
Financement : ARED région Bretagne (50%) - DGA (50%)
Encadrement : Guy Gogniat (50%), Vianney Lapôtre (50%)

Nicolas Gaudin
Années de thèse : depuis 2021
Titre : Développement et évaluation d'un processeur RISC-V robuste face à des attaques par canaux auxiliaires
Financement : LabEx CominLabs - Projet SCRATCHS
Encadrement : Guy Gogniat (25%), Vianney Lapôtre (50%), Pascal Cotret (25%)

William Pensec
Années de thèse : depuis 2021
Titre : Protection d'un processeur avec DIFT contre des attaques physiques
Financement : ARED-PEC (50%) - CDE (50%)
Encadrement : Guy Gogniat (50%), Vianney Lapôtre (50%)

1.5.2 Thèses soutenues

Noura Ait Manssour
Années de thèse : 2019-2023
Titre : Sécurisation matérielle de processeurs embarqués face aux attaques par injection de fautes
Financement : ARED-PEC
Encadrement : Arnaud Tisserand (40%), Vianney Lapôtre (40%), Guy Gogniat (20%)

Ghita Harcha
Années de thèse : 2017-2021
Titre : Introduction d'aléas dans les architectures matérielles pour une contribution à la sécurisation de chiffreurs AES dans un contexte IoT
Financement : ARED-PEC
Encadrement : Philippe Coussy (33%), Vianney Lapôtre (33%), Cyrille Chavet (33%)
Situation : Ingénieure chez Nokia

Maria Mushtaq
Années de thèse : 2016 - 2019
Titre : Software-based detection and mitigation of microarchitectural attacks on Intel's x86 architecture
Financement : ARED région Bretagne / CDE UBS
Encadrement : Guy Gogniat (50%), Vianney Lapôtre (50%)
Situation : Enseignant-chercheur à IMT Télécom Paris

Cyrielle Feron
Années de thèse : 2015 - 2018
Titre : PAnTHErS : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe
Financement : ENSTA Bretagne / Brest métropole
Encadrement : Loïc Lagadec (50%), Vianney Lapôtre (50%)
Situation : Ingénieure chez DGA-MI, Bruz, France

Vincent Migliore
Années de thèse : 2014 - 2017
Titre : Cybersécurite matérielle et conception de composants dédiés au calcul homomorphe
Financement : DGA
Encadrement : Guy Gogniat (30%), Vianney Lapôtre (30%), Arnaud Tisserand (20%), Caroline Fontaine (20%)
Situation : Maître de Conférences à INSA Toulouse

Maria Méndez Real
Années de thèse : 2014 - 2017
Titre : Spatial Isolation against Logical Cache-based Side-Channel Attacks in Many-Core Architectures
Financement : ANR - projet TSUNAMY
Encadrement : Guy Gogniat (50%), Vianney Lapôtre (50%)
Situation : Maître de Conférences à Polytech, Université de Nantes

1.6 Publications

Mon activité scientifique menée depuis 2010 a conduit aux publications suivantes : 11 revues scientifiques internationales, 1 revue scientifique nationale, 27 publications en conférences internationales, 5 publications en conférences nationales. Cela représente 42 publications scientifiques.

Publications en revues (12)	**revues scientifiques internationales** - IEEE Transactions on Very Large Scale Integration Systems (TVLSI) - IEEE Transactions on Computers (TC) - EURASIP Journal on Advances in Signal Processing - ACM Transactions on Embedded Computing Systems (TECS) - Elsevier Information Systems - Elsevier Journal of Information Security and Applications (JISA) - Spinger Annals of Telecommunications - IEEE Access **revues scientifiques nationales** - Technique et Science Informatiques (TSI)
Publications en conférences (32)	**conférences internationales avec comité de lecture** FPL, FPT, ASP-DAC, ISCAS, SECRYPT, NTMS, ISVLSI, AsianHOST, CNS, DSD, NEWCAS, ReConFig, RSP, ReCoSoC ICECS, GIIS, SAMOS, PDP, DDECS **conférences nationales avec comité de lecture** RESSI, Compas, Gretsi, SYMPA

2 Résumé des activités de recherche

2.1 Contexte

Mon premier contact avec le travail de recherche académique a eu lieu en 2009/2010 durant ma seconde année de Master. En particulier, j'ai réalisé mon stage de Master au sein du laboratoire Lab-STICC à Lorient. Durant ce stage, j'ai mené des travaux exploratoires concernant l'intégration des techniques de prédiction de branchements dans un flot de synthèse de haut niveau.

J'ai ensuite poursuivi mon cursus par 3 années de doctorat entre 2010 et 2013. Mes travaux de thèse se sont focalisés sur la problématique de reconfiguration dynamique d'une plateforme multiprocesseur pour le turbo-décodage dans un contexte multimode et multistandard. Ces travaux ont abouti au développement de mécanismes permettant la reconfiguration en temps réel avec une granularité au niveau trame. La plateforme proposée est donc capable de traiter plusieurs flux d'informations s'appuyant sur des standards de communication différents en respectant les débits cibles.

J'ai intégré en novembre 2013 le laboratoire LIRMM de Montpellier. Durant 10 mois, j'ai travaillé dans le cadre du projet européen Mont-blanc [1]. Dans le cadre de ce projet, j'ai participé à l'évaluation de technologie issues du monde de l'embarqué pour la construction de supercaculateurs dont l'efficacité énergétique permettrait d'atteindre l'exaflops [2].

En septembre 2014, j'ai réintégré le laboratoire Lab-STICC au sein de l'Université Bretagne Sud à Lorient. Mon activité de recherche s'est alors orientée autour des thèmes de la cybersécurité et des systèmes embarqués. Mes activités, qui se déroulent maintenant au sein de l'équipe ARCAD, peuvent être positionnées selon trois axes de recherche illustrés par la figure 1 : *la sécurité des processeurs embarqués, la protection face aux attaques exploitant la microarchitecture et l'implémentation de fonctions cryptographiques.* Ces trois axes visent à proposer des solutions originales pour sécuriser des systèmes embarqués faces aux attaques logiques et physiques. Ces axes de recherche seront présentés dans la section 2.4. Ces travaux ont été abordés dans le cadre de différents projets collaboratifs de recherche (ANR Tsunamy, CominLabs Hardblare et SCRATCHS, PEPS INS3PECT,...) et de travaux pré et post doctoraux (2 post-docs, 9 thèses, 1 ingénieur, 7 stages).

Dans la suite, un aperçu rapide des activités de recherche ainsi que des travaux menés dans ces thématiques est proposé.

2.2 Activités de recherche "doctorale"

2.2.1 Master : Introduction de la prédiction de branchement dans la synthèse de haut niveau

Mon expérience dans le domaine de la conception de systèmes numériques a débuté lors de mon stage de seconde année de Master (2010) effectué au sein de l'équipe Méthodes et Outils pour la conception de Circuits et Systèmes (MOCS) du laboratoire Lab-STICC de l'Université Bretagne Sud sous l'encadrement du Professeur Philippe Coussy et de Cyrille Chavet (Maître de Conférences).

Durant ces travaux exploratoires, nous avons développé une approche permettant de combiner la spéculation d'opérations, la prédiction de branchements et la synthèse de haut niveau. Dans le flot de synthèse proposé, l'application à synthétiser est dans un premier temps compilée pour obtenir un graphe de flot de contrôle et de données. Un graphe de prédiction et de spéculation (GPS), permettant d'évaluer l'intérêt de prédire des branchements et l'intérêt de spéculer les blocs linéaires d'instructions (BB) associés, est ensuite construit. Un BB cible est alors sélectionné et ordonnancé sous contraintes de ressources. Une paire de BB sources, qui seront prédits à

1. `https://www.montblanc-project.eu/`
2. 10^8 opérations en virgule flottante par seconde

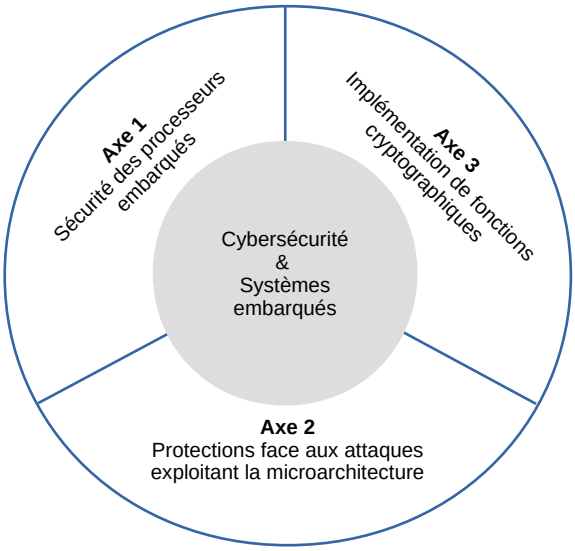


FIGURE 1 – Positionnement des trois axes de recherche

l'exécution du BB cible, est créée. Les opérations de la paire de BB sont alors associées et ordonnancées de façon spéculative dans le BB cible avant d'être assignées sur des opérateurs. Cette étape est répétée tant que tous les BB n'ont pas été ordonnancés. L'architecture de niveau RTL, décrite en VHDL est finalement générée.

Ces travaux de recherche ont conduit à 3 publications scientifiques. 1 conférence internationale [LAP 2013c], 1 revue nationale [LAP 2013b] et 1 conférence nationale [LAP 2011a].

2.2.2 Thèse de doctorat

Mes activités de recherche se sont poursuivies et accélérées à travers la préparation de mon doctorat (2010-2013), au sein du laboratoire Lab-STICC de l'Université Bretagne Sud à Lorient. Mes thèmes de recherche se sont articulés autour des domaines des architectures multiprocesseurs et de la reconfiguration dynamique. Je me suis intéressé à la définition d'une architecture multi-ASIP reconfigurable dynamiquement pour des applications du domaine de la télécommunication.

Thèse de doctorat soutenue le 20 novembre 2013 à l'Université Bretagne Sud, Lorient
Toward dynamically reconfigurable high throughput multiprocessor Turbo decoder in a multimode and multi-standard context
Directeur de thèse : Pr. Guy Gogniat
Mention : très honorable

Composition du jury :

Christophe Jego	IMS, IPB	Rapporteur
Guido Masera	Politecnico di Turino	Rapporteur
Jean-Didier Legat	Univ. Catholique de Louvain	Rapporteur
Édith Beigné	CEA - LETI	Examinateur
Michael Hübner	Ruhr-Universität Bochum	Examinateur
Amer Baghdadi	Telecom Bretagne	Co-directeur
Jean-Philippe Diguet	CNRS	Co-directeur
Guy Gogniat	Université Bretagne-Sud	Directeur de thèse

L'objectif de cette thèse était de développer un récepteur multistandard dynamiquement reconfigurable pour les standards de communication émergents. Pour cela, les travaux se sont

appuyés sur une plateforme multi-ASIP dédiée au Turbo-décodage développée durant des travaux précédents. Toutefois, l'aspect reconfiguration dynamique de la plateforme n'avait pas été abordé. Les travaux de thèse ont été divisés en plusieurs étapes afin d'atteindre cet objectif :

La première étape a été l'étude du processeur DecASIP afin d'optimiser sa conception dans le cadre d'un système multiprocesseur reconfigurable. Cette étape a donné lieu à une nouvelle spécification intégrant une réorganisation du stockage des paramètres de configuration. Cette première contribution a permis d'optimiser les performances de reconfiguration du DecASIP. Une nouvelle implémentation du DecASIP optimisé a également été proposée.

La seconde étape a eu pour but de définir une infrastructure de communication dédiée à la reconfiguration. Cette deuxième contribution a permis d'optimiser le chargement des nouvelles configurations et le contrôle des DecASIP. Pour cela, une approche basée sur une architecture de bus unidirectionnel pipeliné de faible complexité et offrant des mécanismes de multicast et de broadcast a été proposée. Cette solution permet le transfert d'une configuration pour 128 processeurs avec une latence inférieure à la microseconde.

Enfin, la dernière étape des travaux de thèse a été l'étude d'une politique de management de la plateforme afin d'adapter ses paramètres en fonction des données recueillies sur l'environnement et sur l'application exécutée. Cette dernière contribution a abouti au développement d'une approche permettant de supporter la reconfiguration dynamique de la plateforme dans le cas de scénarios à fortes contraintes de débits et de taux d'erreur binaire où chaque trame ou groupe de trames de données est associé à une configuration particulière.

L'ensemble des travaux de recherche menés durant ma thèse entre 2010 et 2013 ont conduit à 10 publications scientifiques : 2 revues internationales [LAP 2016a] [LAP 2017a], 6 conférences internationales [LAP 2013d] [LAP 2013e] [LAP 2013f] [LAP 2013g] [MUR 2013a] [LAP 2012a], 1 conférence nationale [LAP 2013h] et une communication internationale invitée [LAP 2014a].

2.3 Activités de recherche en tant que chercheur contractuel

Avant d'être nommé Maître de Conférences à l'Université de Bretagne Sud, j'ai été chercheur contractuel CNRS au sein du laboratoire LIRMM de Montpellier de novembre 2013 à Août 2014. Durant cette année j'ai poursuivi mon activité de recherche au sein du projet européen Mont-blanc [3]. J'ai notamment participé au co-encadrement de thèse de Rafael Fraga Garibotti, d'Anastasiia Butko et d'Anelise Kologeski. De plus, j'ai encadré Charles Effiong dans le cadre de son stage de seconde année de Master.

Ces travaux de recherche ont conduit à 3 publications scientifiques dans des conférences internationales [BUT 2015a] [EFF 2015a] [KOL 2014a].

2.4 Activités de recherche en tant que Maître de Conférences

Je suis Maître de Conférences au sein de l'Université Bretagne Sud, sur le site de Lorient, depuis septembre 2014. J'ai d'abords intégré l'équipe de recherche Méthodes et Outils pour les Circuits et Systèmes (MOCS) du laboratoire Lab-STICC. Puis, suite à une restructuration du laboratoire en 2020, j'ai intégré l'équipe architectures matérielles et outils de CAO (ARCAD) dont je suis aujourd'hui le responsable. Mon activité de recherche se focalise les thèmes de la cybersécurité et des systèmes embarqués selon trois axes de recherche présentés dans les sections suivantes.

2.4.1 Axe 1 : Sécurité des processeurs embarqués

L'objectif de cet axe est de concevoir des architectures de processeurs intégrant, à la conception, des protections contre les attaques logiques et physiques. L'objectif est de proposer des

3. `https://www.montblanc-project.eu/`

solutions s'appuyant sur des éléments matériels afin de minimiser le coût lié à la sécurité en terme de performance et d'énergie. Plusieurs travaux contribuent à cet axe :

— Protection d'un processeur avec DIFT [4] contre des attaques physiques (2021 - en cours)

— Rejeu matériel d'instructions face aux attaques par injection de fautes (2019 - 2023)

— Co-conception logicielle/matérielle pour le Dynamic Information Flow Tracking (2015 - 2019)

Les travaux menés au sein de cet axe de recherche ont conduit à 6 communications scientifiques réparties comme suit : 5 conférences internationales [AIT 2022] [WAH 2018a], [WAH 2018b] [WAH2017a], [WAH 2016a] et 1 conférence nationale [WAH 2017b].

2.4.2 Axe 2 : Protections face aux attaques exploitant la microarchitecture

Cet axe de recherche s'intéresse au développement de protections face aux attaques exploitant la microarchitecture. Plus particulièrement, les travaux menés se focalisent sur les attaques s'appuyant sur l'observation de variations temporelles durant l'exécution de codes logiciels pour extraire des informations sensibles manipulées par un programme victime. Ces travaux s'intéressent particulièrement aux fuites d'information via les mémoires caches. Plusieurs contributions participent à cet axe de recherche :

— Conception conjointe d'un processeur et du compilateur associé pour lutter contre les attaques par canaux auxiliaires exploitant les variations temporelles durant l'exécution de code (2021 - en cours)

— Détection logicielle d'attaques exploitant des canaux auxiliaires exploitant les mémoires caches (2016 - 2019)

— Isolation spatiale contre les attaques par canaux auxiliaires exploitant les mémoires caches au sein des architectures many-core (2014 - 2017)

Les travaux menés au sein de cet axe de recherche ont conduit à 13 communications scientifiques réparties comme suit : 5 revues internationales [MUK 2020a] [MUS 2020a] [AKR 2020a] [MUS 2020b] [MEN 2018a], 7 conférences internationales [MUS 2019a], [MUS 2018a], [MUS 2018b], [MUS 2018c], [MEN 2016a], [MEN 2016b], [MEN 2016c] et 1 conférences nationale, [MEN 2016d].

2.4.3 Axe 3 : Implémentation de fonctions cryptographiques

Cet axe de recherche se focalise sur l'implémentation de fonctions cryptographiques. L'objectif de cet axe est de proposer des solutions pour l'implémentation matérielle sous contraintes de performance temporelle, de surface, d'efficacité énergétique et de sécurité. Plusieurs contributions ont été apportées au sein de cet axe :

— Introduction d'un mécanisme de shuffling matériel pour l'algorithme de chiffrement AES dans un contexte IoT (2017 - 2021)

— Réalisation d'un outil de prototypage et d'aide à l'exploration de paramètres des schémas de chiffrement homomorphe (2015 - 2018)

— Co-conception logicielle / matérielle pour l'accélération de schémas de chiffrement homomorphe (2014 - 2017)

Les travaux menés au sein de cet axe de recherche ont conduit à 9 communications scientifiques réparties comme suit : 3 revues internationales [FER 2020a], [MIG 2018a], [MIG 2017a], 5 conférences internationales [HAR 2020a], [FER 2018a], [FER 2017a], [MIG 2016a], [MIG 2015a] et 1 conférence nationale [MIG 2016b].

4. Dynamic Information Flow Tracking

2.5 Encadrement de travaux de recherche

Cette partie concerne les activités d'encadrement de travaux de recherche, à savoir :

- Encadrement de doctorants
- Encadrement de post-doctorants
- Encadrement d'ingénieurs
- Encadrement de stagiaires de Master recherche

2.5.1 Encadrement de doctorants

Depuis 2014 j'ai co-encadré 9 thèses. Six thèses ont été soutenues, respectivement en septembre 2017 (2 soutenances), novembre 2018, septembre 2019, juillet 2021 et janvier 2022. Trois thèses sont actuellement en cours.

- Maria Méndez Real - contrat ANR

Spatial Isolation against Logical Cache-based Side-Channel Attacks in Many-Core Architectures
Thèse de doctorat soutenue le 20 septembre 2017
En co-encadrement avec Guy Gogniat (50%)
Résumé : *L'évolution technologique ainsi que l'augmentation incessante de la puissance de calcul requise par les applications font des architectures "many-core" la nouvelle tendance dans la conception des processeurs. Ces architectures sont composées d'un grand nombre de ressources de calcul (des centaines ou davantage) ce qui offre du parallélisme massif et un niveau de performance très élevé. En effet, les architectures many-core permettent d'exécuter en parallèle un grand nombre d'applications, venant d'origines diverses et de niveaux de sensibilité et de confiance différents, tout en partageant des ressources physiques telles que des ressources de calcul, de mémoire et de communication. Cependant, ce partage de ressources introduit également des vulnérabilités importantes en termes de sécurité. En particulier, les applications sensibles partageant des mémoires cache avec d'autres applications, potentiellement malveillantes, sont vulnérables à des attaques logiques de type canaux cachés basées sur le cache. Ces attaques, permettent à des applications non privilégiées d'accéder à des informations secrètes sensibles appartenant à d'autres applications et cela malgré des méthodes de partitionnement existantes telles que la protection de la mémoire et la virtualisation. Alors que d'importants efforts ont été faits afin de développer des contremesures à ces attaques sur des architectures multicoeurs, ces solutions n'ont pas été originellement conçues pour des architectures many-core récemment apparues et nécessitent d'être évaluées et/ou revisitées afin d'être applicables et efficaces pour ces nouvelles technologies. Dans ce travail de thèse, nous proposons d'étendre les services du système d'exploitation avec des mécanismes de déploiement d'applications et d'allocation de ressources afin de protéger les applications s'exécutant sur des architectures many-core contre les attaques logiques basées sur le cache. Plusieurs stratégies de déploiement sont proposées et comparées à travers différents indicateurs de performance. Ces contributions ont été implémentées et évaluées par prototypage virtuel basé sur SystemC et sur la technologie "Open Virtual Platforms" (OVP).*
Composition du jury : Nele Mentens (Professeure associée, KU Leuven, rapporteur), Eduardo De La Torre (Professeur associé, Universidad Politécnica de Madrid (UPM), rapporteur), Gilles Sassatelli (Directeur de recherche CNRS, rapporteur), Lilian Bossuet (Professeur des Universités, Université Jean Monnet, président de jury), Khurram Bhatti (Professeur associé, Technology University (TU) Lahore, examinateur), Diana Goehringer (Professeure, Technische Universitaet Dresden, examinatrice), Vianney Lapotre (Maître de Conférences, Université Bretagne Sud, examinateur), Guy Gogniat (Professeur des Universités, Université Bretagne Sud, directeur de thèse).

Situation : Maîtresse de Conférences à Polytech, Université de Nantes

- Vincent Migliore - Bourse DGA

Cybersécurite matérielle et conception de composants dédiés au calcul homomorphe
Thèse de doctorat soutenue le 26 septembre 2017
En co-encadrement avec Guy Gogniat (30%), Arnaud Tisserand (20%) et Caroline Fontaine (20%)
Résumé : *L'émergence d'internet et l'amélioration des infrastructures de communication ont considérablement encouragé l'explosion des flux d'in- formations au niveau mondial. Cette évolution a été accompagnée par l'apparition de nouveaux besoins et de nouvelles attentes de la part des consommateurs. Communiquer avec ses proches ou ses collaborateurs, stocker des documents de travail, des fichiers multimédia, utiliser des services innovants traitant nos documents personnels, tout cela se traduit immanquablement par le partage, avec des tiers, d'informations potentiellement sensibles. Ces tiers, s'ils ne sont pas de confiance, peuvent réutiliser à notre insu les données sensibles que l'on leur a confiées. Dans ce contexte, le chiffrement homomorphe apporte une bonne solution. Il permet de cacher aux yeux des tiers les données qu'ils sont en train de manipuler. Cependant, à l'heure actuelle, le chiffrement homomorphe reste complexe. Pour faire des opérations sur des données de quelques bits (données en clair), il est nécessaire de manipuler des opérandes sur quelques millions de bits (données chiffrées). Ainsi, une opération normalement simple devient longue en termes de temps de calcul. Dans cette étude, nous avons cherché à rendre le chiffrement ho- momorphe plus pratique en concevant un accélérateur spécifique. Nous nous sommes basés sur une approche de type co-conception logicielle/matérielle utilisant l'algorithme de Karatsuba. En particulier, notre approche est compatible avec le batching, qui permet de stocker plusieurs bits d'informations dans un même chiffré. Notre étude démontre que le batching peut être implémenté sans surcoût important comparé à l'approche sans batching, et permet à la fois de réduire les temps de calcul (calculs effectués en parallèle) et de réduire le rapport entre la taille des données chiffrées et des données en clair.*
Composition du jury : Guy Gogniat (Professeur des Universités, Université Bretagne Sud, directeur de thèse), (Arnaud Tisserand (Directeur de Recherche CNRS, co-directeur de thèse), Caroline Fontaine (Chargée de Recherche CNRS, co-directrice de thèse), Régis Leveugle (Professeur des Universités, Grenoble INP - Phelma, rapporteur), Nicolas Sendrier (Directeur de recherche, INRIA, rapporteur), Renaud Sirdey (Directeur de recherche, CEA, examinateur), Russell Tessier (Professeur, University of Massachusetts (USA), examinateur), Vianney Lapôtre (Maître de Conférences, Université Bretagne Sud, examinateur)
Situation : Maître de Conférences à INSA Toulouse

- Cyrielle Feron - Bourse ENSTA Bretagne / Brest métropole

PAnTHErS : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe
Thèse de doctorat soutenue le 14 novembre 2018
En co-encadrement avec Loïc Lagadec (50%)
Résumé : *Le chiffrement homomorphe est un système de cryptographie permettant la manipulation de données chiffrées. Cette propriété offre à un utilisateur la possibilité de déléguer des traitements sur ses données privées, à un tiers non fiable sur un serveur distant, sans perte de confidentialité.Bien que les recherches sur l'homomorphe soient, à ce jour, encore récentes, de nombreux schémas de chiffrement ont été mis au point. Néanmoins, ces schémas souffrent de quelques inconvénients, notamment, de temps d'exécution particulièrement longs et de coûts mémoire importants. Ces limitations rendent difficile la comparaison des schémas afin de déterminer lequel serait le plus adapté pour une application donnée, c'est-à-dire le moins coûteux en temps et en mémoire.Ce manuscrit présente PAnTHErS, un outil rassemblant plusieurs fonctionnalités permettant de répondre à la problématique citée ci-dessus. Dans l'outil PAnTHErS,*

les schémas de chiffrement homomorphe sont tout d'abord représentés dans un format commun grâce à une méthode de modélisation. Puis, une analyse théorique estime, dans le pire cas, la complexité algorithmique et le coût mémoire de ces schémas en fonction des paramètres d'entrée fournis. Enfin, une phase de calibration permet la conversion des analyses théoriques en résultats concrets : la complexité algorithmique est convertie en un temps d'exécution estimé en secondes et le coût mémoire en une consommation estimée en mébioctets.Toutes ces fonctionnalités associées ont permis la réalisation d'un module d'exploration qui, à partir d'une application, sélectionne les schémas ainsi que les paramètres d'entrée associés produisant des temps d'exécution et coûts mémoire proches de l'optimal.
Composition du jury : Lilian Bossuet (Professeur des Universités, Université Jean-Monnet, rapporteur), Renaud Sirdey (Directeur de recherche au CEA, rapporteur), Régis Leveugle, (Professeur, INP Grenoble, président du jury), Caroline Fontaine (Chargée de Recherche au CNRS, examinatrice), Loïc Lagadec (Professeur, ENSTA Bretagne, directeur de thèse), Vianney Lapôtre (Maître de Conférences, Université Bretagne Sud, examinateur)
Situation : Ingénieure chez DGA-MI, Bruz, France

- Maria Mushtaq - ARED région Bretagne / CDE Université Bretagne Sud

Software-based detection and mitigation of microarchitectural attacks on Intel's x86 architecture
Thèse de doctorat soutenue le 16 septembre 2019
En co-encadrement avec Guy Gogniat (50%)
Résumé : *Les attaques par canaux cachés basées sur les accès aux mémoires caches constituent une sous-catégorie représentant un puissant arsenal permettant de remettre en cause la sécurité d'algorithmes cryptographiques en ciblant leurs implémentations. Malgré de nombreux efforts, les techniques de protection contre ces attaques ne sont pas encore assez matures. Ceci est principalement dû au fait que la plupart des techniques ne protègent généralement pas contre tous les scénarii d'attaques. De plus, ces solutions peuvent impacter fortement les performances des systèmes. Cette thèse propose des arguments en faveur du renforcement de la sécurité et de la confidentialité dans les systèmes informatiques modernes tout en conservant leurs performances. Pour cela, la thèse développe une protection basée sur les besoins, qui permettent au système d'exploitation d'appliquer uniquement des mesures de protection après la détection des attaques. Ainsi, la détection peut servir de première ligne de défense. Cependant, pour que la stratégie de protection basée sur la détection soit efficace, il faut que cette dernière soit fiable, n'impacte que faiblement les performances et couvre un large spectre d'attaques avant que ces dernières atteignent leur but. Dans cette optique, cette thèse propose un cadre complet pour la protection basée sur la détection d'un ensemble d'attaques exploitant les mémoires caches lors de l'exécution sous des conditions de charge variables du système. De plus, la thèse propose de coupler l'utilisation du principe de détection avec un mécanisme de protection intégré au système d'exploitation Linux. Bien que le mécanisme de protection proposé soit appliqué à Linux, la solution est extensible à d'autres systèmes d'exploitation. Cette thèse démontre que la sécurité et la confidentialité doivent être pris en compte au niveau système et que les solutions de protection doivent adopter une approche holistique.*
Composition du jury : pascal Benoit (Maître de Conférences, Université Montpellier 2, rapporteur), Karine Heydemann (Maître de Conférences, Sorbonne université, rapporteur), Lilian Bossuet (Professeur, Université Jean-Monnet, président du jury), Sylvain Guilley (Professeur, TELECOM-ParisTtech, examinateurs), Gilles Sassatelli, (Directeur de recherche CNRS, examinateur), Guy Gogniat (Professeur des Universités, université Bretagne Sud, directeur de thèse), Vianney Lapôtre (Maître de Conférences, université Bretagne Sud, examinateur), Muhammad Khurram Bhatti (Professeur associé, information Technology university Lahore, examinateur)
Situation : Enseignant-chercheur à IMT Télécom SudParis

- Ghita Harcha - Bourse ARED-PEC

Introduction d'aléas dans les architectures matérielles pour une contribution à la sécurisation de chiffreurs AES dans un contexte IoT
Thèse de doctorat soutenue le 13 juillet 2021
En co-encadrement avec Philippe Coussy (33%) et Cyrille Chavet (33%)

Résumé : *Nous vivons dans un monde où l'information et l'échange de données sont devenus des éléments clefs de nos économies. Il faut ajouter à cela l'explosion et la diffusion rapide de ce que l'on appelle l'internet des objets (IoT, Internet of Things) à tous les niveaux de nos sociétés et dans nos vies, tant professionnelles que personnelles. Il s'agit là de systèmes embarqués communicants très contraints en taille et en énergie, déjà largement déployés. Toutefois, ces derniers présentent de nombreuses vulnérabilités et de ce fait font partie des cibles privilégiées pour des attaques malveillantes. C'est pourquoi, ces dispositifs s'accompagnent de plus en plus de systèmes de chiffrement. Malheureusement, leurs implémentations peuvent elles-mêmes être sujettes à des failles.*
Dans cette thèse nous nous intéressons à la sécurisation d'une architecture de chiffrement AES face à des attaques par canaux cachés, notamment les attaques dites par "observation de consommation de puissance". Le domaine de l'IoT étant ciblé, des architectures d'AES faible coût sont visées et l'objectif est de minimiser l'impact en termes de surface, débit, latence et consommation. L'approche proposée consiste à adjoindre à un composant AES un module de génération d'aléa. Dans ce contexte, plusieurs solutions architecturales ont étés étudiées : nombre de permutations et type de d'informations permutées. La robustesse des différents architectures face à différentes attaques de l'état de l'art est évaluée. Les surcouts induits par le composant de brassage sont quantifiés et les effets des options de synthèse sont étudiés.
Les résultats montrent qu'avec notre modèle architectural le plus sûr (et donc le plus complexe et le plus coûteux) aucun octet de la clef de chiffrement n'est révélé après un million d'échantillons mesurés sur FPGA. Cet apport en sécurité a un coût, contenu au regard des solutions présentes dans la littérature : un débit divisé d'un facteur 2,3 ; une réduction de 11% de la fréquence max ; un surcoût matériel équivalent à environ 3,9 fois l'architecture d'origine.
Composition du jury : Cécile Belleudy, (Maîtresse de Conférences – HDR, Université Côte d'azur, rapporteur), Roselyne Chotin (Maîtresse de Conférences – HDR, Sorbonne Université, rapporteur), Lilian Bossuet (Professeur des Universités, Université Jean Monnet, président du jury), Emmanuel Casseau (Professeur des Universités, Université Rennes 1, examinateur), Christophe Jego (Professeur des Universités, Bordeaux INP, examinateur), Philippe Coussy (Professeur des Universités, Université Bretagne Sud, directeur de thèse), Vianney Lapôtre (Maître de conférence, Université Bretagne Sud, examinateur), Cyrille Chavet (Maître de conférence, Université Bretagne Sud, examinateur)
Situation : Ingénieure chez Nokia

- Noura Ait Manssour (2019-2022) - Bourse PEC Région Bretagne / DGA

Sécurisation matérielle de processeurs embarqués face aux attaques par injection de fautes
Thèse de doctorat soutenue le 13 janvier 2023
En co-encadrement avec Arnaud Tisserand (40%) et Guy Gogniat (20%)

Résumé : *Les processeurs embarqués peuvent faire l'objet d'attaques physiques en raison de la proximité entre l'attaquant et le circuit. Les attaques par injection de fautes (FIA) exploitent des perturbations du circuit pour révéler des données secrètes ou contourner des dispositifs de sécurité. Il existe plusieurs méthodes de protection contre les FIA : correction/détection d'erreurs, vérification des propriétés fonctionnelles, redondance, randomisation, etc. En logiciel, la duplication et la triplication d'instructions sont faciles à utiliser pour sécuriser des codes critiques mais*

entraînent des surcoûts importants en temps d'exécution et taille de code. De plus, les protections logicielles prennent rarement en compte les détails d'implémentation du matériel, comme le pipeline du processeur, et peuvent ne pas être aussi efficaces que prévu. Nous proposons un support matériel pour le rejeu d'instructions dans un processeur RISC élémentaire. Il consiste en une petite extension du jeu d'instructions (une nouvelle instruction), quelques éléments de détection et de vote (principalement des registres et des comparateurs) et de légères modifications du contrôle du processeur. Nous explorons différentes configurations d'éléments de protection internes pour le rejeu matériel. Le surcoût en surface du cœur est de 30rejeu matériel réduit de manière significative le temps d'exécution et la taille du code par rapport à une protection purement logicielle.
Composition du jury : Karine Heydemann, (Maîtresse de Conférences – HDR, Sorbonne Université, rapporteur), Pascal Benoit (Maître de Conférences – HDR, Université de Montpellier, rapporteur), Vincent Beroulle (Professeur des Universités, Grenoble INP, président du jury), Arnaud Tisserand (Directeur de Recherche CNRS, directeur de thèse), Vianney Lapôtre (Maître de conférence, Université Bretagne Sud, examinateur), Guy Gogniat (Professeur des Universités, Université Bretagne Sud, co-directeur de thèse)

- William Pensec (2021-2024) - Bourse PEC Région Bretagne / CDE UBS

Protection d'un processeur avec DIFT contre des attaques physiques
Soutenance prévue en 2024
En co-encadrement avec Guy Gogniat (50%)

- Nicolas Gaudin (2021-2024) - Contrat LabEx CominLabs

Développement et évaluation d'un processeur RISC-V robuste face à des attaques par canaux auxiliaires
Soutenance prévue en 2024
En co-encadrement avec Guy Gogniat (25%) et Pascal Cotret (25%)

- Hongwei Zhao (2022-2025) - Bourse ARED Région Bretagne / DGA

Architecture de communication sécurisée d'un SoC vis-à-vis des attaques physiques et logiques
Soutenance prévue en 2025
En co-encadrement avec Guy Gogniat (50%)

La Table 2 résume l'ensemble des thèses réalisées en co-encadrement. Différentes sources de financement ont été utilisées afin de financer les doctorants (Contrats ANR et LabEx CominLabs, bourses région ARED, Direction générale de l'armement (DGA), et CDE de l'Université Bretagne Sud).

2.5.2 Encadrement de post-doctorants

- Arnab kumar Biswas (2018-2019) - contrat LabEx CominLabs

Optimisation de l'architecture mémoire d'un co-processeur pour le DIFT
Février 2018 / November 2019

- Kamel Aizi (2022-2024) - bourse DGA

Évaluation de moniteurs de sécurité face à des attaques physique exploitant des canaux auxiliaires
Décembre 2022 / Décembre 2024

TABLE 2 – Résumé des co-encadrements de thèses et du devenir des étudiants.

	14/15	15/16	16/17	17/18	18/19	19/20	20/21	21/22	22/23	23/24	24/25
Maria Méndez Real (ANR)	Spatial Isolation against Logical Cache-based Side-Channel Attacks in Many-Core Architectures			Maîtresse de conférences à Polytech, Université de Nantes							
Vincent Migliore (DGA)	Cybersécurité matérielle et conception de composants dédiés au calcul homomorphe			Maître de Conférences à INSA Toulouse							
Cyrielle Feron (ENSTA Bretagne / Brest métropole)		PAnTHErS : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe			Ingénieure à DGA-MI, Bruz						
Maria Mushtaq (ARED/CDE)			Software-based detection and mitigation of microarchitectural attacks on Intel's x86 architecture			Enseignant-chercheur à IMT Télécom SudParis					
Ghita Harcha (ARED-PEC)				Introduction d'aléas dans les architectures matérielles pour une contribution à la sécurisation de chiffreurs AES dans un contexte IoT			Ingénieure chez Nokia, Paris				
Noura Ait Manssour (ARED-PEC)						Protection d'un processeur face à des attaques logicielles					
William Pensec (ARED-PEC/CDE)								Protection d'un processeur avec DIFT contre des attaques physiques			
Nicolas Gaudin (LabEx CominLabs)								Développement et évaluation d'un processeur RISC-V robuste face à des attaques par canaux auxiliaires			
Hongwei Zhao (ARED/DGA)									Architecture de communication sécurisée d'un SoC vis-à-vis des attaques physiques et logiques		

2.5.3 Encadrement d'ingénieurs

- Cédric Seguin (2016-2017) - contrat PEPS SISC INS2I - HomCrypt

Développement d'un démonstrateur pour l'accélération matérielle du chiffrement homomorphe
Septembre 2016 – Mars 2017

2.5.4 Encadrement de stagiaires de Master

J'ai encadré 7 étudiants en Master sur des sujets relatifs à la conception des systèmes embarqués et à la mise en œuvre de solutions de protection pour les systèmes multiprocesseurs.

- Robin Danilo

Introduction de la prédiction de branchement dans la synthèse de haut niveau
Master SIAM, Lorient, Année universitaire 2011/2012

- Charles Effiong

Exploration des performances d'un NoC 3D intégrant des liens verticaux défaillants
Master systèmes embarquées, Brest, Année universitaire 2013/2014

- Thomas Toublanc

Implémentation FPGA d'un système multiprocesseur intégrant un système d'exploitation
Master SIAM, Lorient, Année universitaire 2014/2015

TABLE 3 – Résumé des encadrements des stages de Master recherche.

	11/12	12/13	13/14	14/15	15/16	16/17	17/18	21/22
Robin Danilo (Master Lorient)	Introduction de la prédiction de branchement dans la synthèse de haut niveau							
Charles Effiong (Master Brest)			Exploration des performances d'un NoC 3D intégrant des liens verticaux défaillants					
Thomas Toublanc (Master Lorient)				Implémentation FPGA d'un système multiprocesseur intégrant un système d'exploitation				
Djelar Esperance Asngar (Master Lorient)				Analyse et comparaison des simulateurs GEM5 et OVPsim				
Samy Rida (Master Lorient)	Implémentation de contremesures face aux attaques par canaux auxiliaires via les mémoires caches							
Jérémy Bricq (Master Bruxelles)	Détection des attaques par canaux auxiliaires via les mémoires caches au niveau OS							
Kévin Queneherve (Master Lorient)	Injection de fautes par glitch d'horloge ciblant un processeur RISC-V							

- Djelar Esperance Asngar

Analyse et comparaison des simulateurs GEM5 et OVPsim
Master SIAM, Lorient, Année universitaire 2014/2015

- Samy Rida

Implémentation de contremesures face aux attaques par canaux auxiliaires via les mémoires caches
Master CSSE, Lorient, Année universitaire 2017/2018

- Jérémy Bricq

Détection des attaques par canaux auxiliaires via les mémoires caches au niveau OS
Master Cybersécurité, Bruxelles, Année universitaire 2017/2018

- Kévin Queneherve

Injection de fautes par glitch d'horloge ciblant un processeur RISC-V
Master CSSE, Lorient, Année universitaire 2021/2022

La Table 3 résume l'ensemble des stages de Master recherche dont j'ai assuré l'encadrement.

2.6 Responsabilités scientifiques

Cette partie concerne les responsabilités scientifiques, à savoir :

- Participation à des jurys de thèse
- Participation à un comité de sélection MCF
- Direction / édition d'ouvrages scientifiques
- Participation à des comités de programmes de conférences
- Participation à des comités de lecture de conférences
- Participation à des comités de lecture de journaux nationaux et internationaux
- Participation à des comités d'organisation
- Modération dans des conférences internationales
- Expertise scientifique

2.6.1 Participation à un jury de thèse

- Jonathan Roux

Détection d'intrusion dans des environnements connectés sans-fil par l'analyse des activités radio
Thèse de doctorat soutenue le 2 février 2020

Composition du jury : Lilian Bossuet (Professeur des Universités, Université Jean Monnet, rapporteur), Samia Bouzefrance (Professeure des Universités, CNAM Paris, rapporteur), Grégory BLANC (Enseignant-chercheur, IMT Télécom SudParis, examinateur), Vianney Lapôtre (Maître de Conférences, université Bretagne Sud, examinateur), Ludovic MÉ (Enseignant-chercheur, INRIA, examinateur), Mohamed Kaâniche (Directeur de recherche, CNRS, examinateur), Éric Alata (Maître de Conférences, INSA Toulouse, examinateur), Vincent Nicomette (Professeur des Universités, INSA TOulouse, directeur de thèse)

2.6.2 Participation à un comité de sélection MCF

- Université de Bretagne Occidentale (2018)

2.6.3 Direction / édition d'ouvrages scientifiques

- Éditeur invité du Journal Applied sciences, special issue on Side channel attacks in embedded systems), MDPI, 2021

2.6.4 Participation à des comités de programmes de conférences

Conférences internationales

- Alchemy Thematic track of the International Conference on Computational Science - ICCS (2015, 2016, 2017, 2018, 2019)
- International Symposium on Applied Reconfigurable Computing - ARC (2015, 2016, 2017, 2018)
- Workshop on Cryptography and Security in Computing Systems - CS2 (2015, 2016, 2019)
- IEEE Latin America Symposium on Circuits and System - LASCAS (2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022)
- IEEE International NEWCAS Conference - NEWCAS (2015)
- IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip - MCSoC (2015, 2016, 2017, 2018, 2019, 2021)
- International Workshop on Components and Services for IoT platforms - WCSIoT (2015)
- Baltic Electronics Conference - BEC (2016, 2018)
- Argentine Conference on Micro-Nanoelectronics - EAMTA (2016)
- Conference on PhD Research in Microelectronics and Electronics - PRIME (2017)
- International Conference on Field-Programmable Logic and Applications - FPL (2017, 2018, 2019, 2020, 2021, 2022)
- Symposium on Integrated Circuits and System Design - SBCCI (2018, 2020, 2021)
- Workshop on the Security of Software / Hardware Interfaces - SLIM (2020, 2021, 2022)
- International Conference on Emerging Security Information, Systems and Technologies - SECUWARE (2020)
- IEEE Asia Pacific Conference on Circuits and Systems - APCCAS (2020)

- International Workshop on Security Proofs for Embedded Systems - PROOFS (2022)

Conférence nationale

- Conférence francophone d'informatique en Parallélisme, Architecture et Système - Compas (2022)

2.6.5 Participation à des comités de lecture de conférences

Conférences internationales

- IEEE International Symposium on System-on-Chip - SoC (2013)
- IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM (2013)
- Electronic System Level Synthesis Conference - ESLsyn (2014)
- Colloque africain sur la rechercheen informatique et mathématiques appliquées- CARI (2014)
- International Symposium on Reconfigurable Communication-centric Systems-on-Chip - ReCoSoC (2014)
- Design, Automation and Test in Europe Conference - DATE (2015, 2016, 2018, 2019, 2020)
- IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP (2016, 2017, 2019)
- Great Lakes Symposium on VLSI - GLSVLSI (2016)
- IEEE International Conference on Application-specific Systems, Architectures and Processors - ASAP (2016)
- IEEE Workshop on Signal Processing Systems - SiPS (2017, 2018, 2019, 2020)
- Design Automation Conference - DAC (2017, 2021, 2022)
- IEEE International NEWCAS Conference - NEWCAS (2019)
- Forum on specification & Design Languages - FDL (2019)

Conférence nationale

- Conférence d'informatique en Parallélisme, Architecture et Système - Compas (2016)

2.6.6 Participation à des comités de lecture de journaux internationaux

- Springer Journal of Supercomputing (2016, 2017, 2021)
- Elsevier MICPRO (2016, 2017, 2019, 2020, 2021, 2022)
- IEEE Transactions on Parallel and Distributed Systems - TPDS (2016, 2017)
- IEEE Transactions on Computer-Aided Design - TCAD (2017, 2020, 2021)
- IEEE Transactions on Computers - TC (2018, 2020, 2022)
- IEEE Transactions on Consumer Electronics - TCE (2022)
- ACM Transactions on Embedded Computing Systems - TECS (2019)
- IEEE Transactions on Very Large Scale Integration (VLSI) Systems - TVLSI (2016, 2017 2021, 2022)
- IEEE Embedded Systems Letters (2019, 2022)
- Integration the VLSI Journal, Elsevier (2020)
- IEEE ACCESS (2020)
- Elsevier Journal of Systems Architecture (2022)

2.6.7 Participation à des comités d'organisation

internationaux

- Membre du comité d'organisation local - International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Montpellier, 2014
- PHD forum, Industrial, demo night co-chair - International Conference on Field-Programmable Logic and Applications (FPL), Ghent, 2017
- Membre du comité d'organisation local et Publications and Publicity co-chair - IEEE Workshop on Signal Processing Systems (SiPS), Lorient, 2017
- Membre du comité d'organisation local - Workshop on Cryptographic architectures embedded in logic devices (CryptArchi), Lorient, 2018
- Program co-chair - Alchemy Thematic track of the International Conference on Computational Science, 2019

nationaux

- Membre du comité d'organisation de la Conférence d'informatique en Parallélisme, Architecture et Système - Compas, Lorient, 2016
- Membre du comité d'organisation de l'école thématique architecture des systèmes matériels et logiciels embarqués, et méthodes de conception associées Lorient, 2019
- Co-organisateur de la journée "Sécurité, fiabilité et test des SoC2 : challenges et opportunités dans l'ère de l'Intelligence Artificielle", journée thématique du GDR SOC2, Paris, 2019

2.6.8 Modération dans des conférences internationales

session

- International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Montpellier, 2014
- International Conference on Field-Programmable Logic and Applications (FPL), Ghent, 2017

2.6.9 Expertise scientifique

- AGP - campagne de bourses de thèse du pôle de recherche cyber (2022)
- Agence Nationale de la Recherche - appel a projets générique (2018, 2019, 2021)
- Région Auvergne Rhône-Alpes - Pack Ambition Recherche 2020

2.7 Diffusion des connaissances et publications scientifique

2.7.1 Thèse de doctorat

[LAP 2013a] V. Lapotre, **Toward dynamically reconfigurable high throughput multiprocessor Turbo decoder in a multimode and multi-standard context**, thèse de doctorat, Université de Bretagne Sud, Lorient, Novembre 2013

2.7.2 Articles dans des revues internationales avec comité de lecture

[MUS 2022] M. Mushtaq, M. M. Yousaf, M. K. Bhatti, V. Lapotre, G. Gogniat, **The Kingsguard OS-level mitigation against cache side-channel attacks using runtime detection**, Annals of Telecommunications, Springer, 2022.

[FER 2020a] C. Feron, V. Lapotre and L. Lagadec, **Automated Exploration of Homomorphic Encryption Scheme Input Parameters**, Journal of Information Security and Applications (JISA), 2020. Associate tool : PAnTHErS

[MUK 2020a] MM. A. Mukhtar, M. Mushtaq, M. K. Bhatti, V. Lapotre, G. Gogniat,**FLUSH + PREFETCH : A Countermeasure Against Access-driven Cache-based Side-Channel Attacks**, Journal of Systems Architecture, vol 104, 2020.

[MUS 2020a] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, P. Benoit, **WHISPER A Tool for Run-time Detection of Side-Channel Attacks**, IEEE Access, vol 8, pp. 83871-83900, 2020.

[AKR 2020a] A. Akram, M. Mushtaq, M. K. Bhatti, V. Lapotre, G. Gogniat, **Meet the Sherlock Holmes' od Side Channel Leakage : A Survey of Cache SCA Detection Techniques**, IEEE Access, vol 8, pp. 70836-70860, 2020.

[MUS 2020b] M. Mushtaq, M. A. Mukhtar, V. Lapotre, M. K. Bhatti, G. Gogniat, **Winter is here ! A decade of cache-based side-channel attacks, detection & mitigation for RSA**, Information Systems, vol 92, 2020.

[MIG 2018a] V. Migliore, M. Mendez Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, **Hardware/Software co-Design of an Accelerator for FV Homomorphic Encryption Scheme using Karatsuba Algorithm**, in IEEE Transactions on Computers, vol.37, Issue.3, pp.335-347, 2018.

[MEN 2018a] M. Mendez Réal, P. Wehner, V. Lapotre, D. Göhringer and G. Gogniat, **Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators**, in ACM Transactions on Embedded Computing Systems (TECS), vol. 12, n. 2, Article 55, February 2018.

[LAP 2017a] V. Lapotre, G. Gogniat, A. Baghdadi, J.-P. Diguet, **Dynamic configuration management of a multi-standard and multi-mode reconfigurable multi-ASIP architecture for turbo decoding**, EURASIP Journal on Advances in Signal Processing, 2017.

[MIG 2017a] V. Migliore, C. Seguin, M. Mendez Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, **A High-Speed Accelerator for Homomorphic Encryption using the Karatsuba Algorithm**, ACM Transactions on Embedded Computing Systems (TECS), Volume 16 Issue 5s, 2017

[LAP 2016a] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, M. Hubner, J.-P. Diguet, **A Dynamically Reconfigurable Multi-ASIP Architecture for Multistandard and Multimode Turbo Decoding**, IEEE Transactions on Very Large Scale Integration (VLSI) Systems , vol.24, no.01, pp.383-387, Jan 2016.

2.7.3 Article dans une revue nationale avec comité de lecture

[LAP 2013b] V. Lapotre, P. Coussy, and C. Chavet, **Introduction de la prédiction de branchement dans la synthèse de haut niveau**, Technique et Science Informatiques, no. 2/2013, 281-301, 2013.

2.7.4 Communications internationales avec comité de lecture et actes

[AIT 2022] N. Ait Manssour, V. Lapotre, G. Gogniat, A. Tisserand, **Processor Extensions for Hardware Instruction Replay against Fault Injection Attacks**, in Proc. of 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2022.

[HAR 2020a] G. Harcha, V. Lapotre, C. Chavet, P. Coussy, **Toward Secured IoT Devices : a Shuffled 8-Bit AES Hardware Implementation**, in Proc. of 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020.

[MUS 2019a] M Mushtaq, A Akram, M.K. Bhatti, U. Ali, V Lapotre, G Gogniat, **Sherlock Holmes of Cache Side-Channel Attacks in Intel's x86 Architecture**, in Proc. of IEEE Conference on Communications and Network Security (CNS), 2019.

[WAH 2018a] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, A. Kumar Beswas, G. Gogniat, V. Lapotre, **A novel lightweight hardware-assisted static instrumentation approach for ARM SoC using debug components**, in proc of Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2018.

[WAH 2018b] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, A. Kumar Beswas, G. Gogniat and V. Lapotre, **A small and adaptive coprocessor for information flow tracking in ARM SoCs**, in proc of 2018 International Conference on Reconfigurable Computing and FPGAs (Reconfig), 2018.

[MUS 2018a] M Mushtaq, A Akram, M.K. Bhatti, M. Chaudhry, V Lapotre, G Gogniat, **NIGHTs-WATCH : a cache-based side-channel intrusion detector using hardware performance counters**, in Proc. of 7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP), 2018.

[MUS 2018b] M Mushtaq, A Akram, M.K. Bhatti, R.N. Bin Rais, V Lapotre, G Gogniat, **Run-time Detection of Prime+Probe Side-Channel Attack on AES Encryption Algorithm**, in Proc. of Global Information Infrastructure and Networking Symposium (GIIS), 2018.

[MUS 2018c] M Mushtaq, A Akram, M.K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V Lapotre, G Gogniat, **Machine Learning For Security : The Case of Side-Channel Attack Detection at Run-time**, in Proc. of 25th IEEE International Conference on Electronics (ICECS), 2018.

[FER 2018a] C. Feron, V. Lapotre and L. Lagadec, **Fast Evaluation of Homomorphic Encryption Schemes based on Ring-LWE**, in Proc. of 9th IFIP International Conference on New Technologies, Mobility & Security (NTMS), 2018.

[FER 2017a] C. Feron, V. Lapotre and L. Lagadec, **PAnTHErS : A Prototyping and Analysis Tool for Homomorphic Encryption Schemes**, 14th International Joint Conference on e-Business and Telecommunications - Volume 6 : SECRYPT, ICETE 2017, pages 359-366. 2017

[WAH 2017a] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre and G. Gogniat, **ARMHEx : A hardware extension for DIFT on ARM-based SoCs**, in Proc. of 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.

[WAH 2016a] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre and G. Gogniat, "**Towards a hardware-assisted information flow tracking ecosystem for ARM proces-**

sors, in Proc. of 26th International Conference on Field Programmable Logic and Applications (FPL), 2016.

[MEN 2016a] M. Méndez Real, V. Migliore, V. Lapotre and G. Gogniat, **ALMOS many-core operating system extension with new secure-aware mechanisms for dynamic creation of secure zones**, in Proc. of 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2016.

[MEN 2016b] M. Mendez Réal, V. Migliore, V. Lapotre, G. Gogniat, P. Wehner, and D. Göhringer, **Dynamic Spatially Isolated Secure Zones for NoC-based Many-core Accelerators**, in Proc. of 8th IEEE International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2016.

[MEN 2016c] M. Mendez Réal, V. Migliore, V. Lapotre, G. Gogniat, P. Wehner, J. Rettkowski and D. Göhringer, **MPSoCSim extension : An OVP Simulator for the Evaluation of Cluster-based Multicore and Many-core architectures**, 4rd Workshop on Virtual Prototyping of Parallel and Embedded Systems (ViPES) as part of the International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS), 2016.

[MIG 2016a] V. Migliore, M. Mendez Réal, V. Lapotre, A. Tisserand, C. Fontaine and G. Gogniat, **Fast polynomial arithmetic for Somewhat Homomorphic Encryption operations in hardware with Karatsuba algorithm**, in proc of the 2016 International Conference on Field-Programmable Technology (FPT '16), Jianguo Hotel, Xi'an, China, 2016.

[MIG 2015a] V. Migliore, M. Méndez Real, V. Lapotre, A. Tisserand, C. Fontaine and G. Gogniat, **Exploration of Polynomial Multiplication Algorithms for Homomorphic Encryption Schemes**, in Proc. of 2015 IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2015.

[BUT 2015a] A. Butko, R. Garibotti, L. Ost, V. Lapotre, A. Gamatié, G. Sassatelli and C. J. Adeniyi-Jones, **A Trace-driven Approach for Fast and Accurate Simulation of Manycore Architectures**, 20th Asia and South Pacific Design Automation Conference (ASP-DAC'2015), 2015.

[EFF 2015a] C. Effiong, V. Lapotre, A. Gamatié, G. Sassatelli, A. Todri and K. Latif, **On the Performance Exploration of 3D NoCs with Resistive-Open TSVs**, in Proc. of 2015 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2015.

[KOL 2014a] A. Kologeski, F. Lima Kastensmidt, V. Lapotre, A. Gamatié, G. Sassatelli, A. Todri-Sanial, **Performance Exploration of Partially Connected 3D NoCs under Manufacturing Variability**, in Proc. of 12th IEEE International New Circuits and Systems Conference (NEWCAS), 2014.

[LAP 2013c] V. Lapotre, P. Coussy, C. Chavet, H. Wouafo, and R. Danilo, **Dynamic branch prediction for high-level synthesis**, in 23rd International Conference on Field Programmable Logic and Applications (FPL), 2013.

[LAP 2013d] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, M. Huebner, J.-P. Diguet, **Stopping-free dynamic configuration of a multi-asip turbo decoder**, in Proc. of 2013 16th Euromicro Conference on Digital System Design (DSD), 2013.

[LAP 2013e] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, M. Huebner, J.-P. Diguet, **A reconfigurable multi-standard asip-based turbo decoder for an efficient dynamic reconfiguration in a multi-asip context**, in Proc. of 2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2013.

[LAP 2013f] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, J.-P. Diguet, J.-N. Bazin, M. Huebner, **Optimizations for an efficient reconfiguration of an asip-based turbo decoder**, in Proc. of 2013 IEEE International Symposium on Circuits and Systems (ISCAS), 2013.

[LAP 2013g] V. Lapotre, M. Huebner, G. Gogniat, P. Murugappa, A. Baghdadi, J.-P. Diguet, **An efficient on-chip configuration infrastructure for a flexible multi-asip turbo decoder architecture**, in Proc. of 8th International Workshop on Reconfigurable Communicationcentric Systems-on-Chip (ReCoSoC), 2013.

[MUR 2013a] P. Murugappa, V. Lapotre, A. Baghdadi, M. Jezequel, **Rapid Design and Prototyping of a Reconfigurable Decoder Architecture for QC-LDPC Codes**, in Proc. Of 2013 IEEE International Symposium on Rapid System Prototyping (RSP), 2013.

[LAP 2012a] V. Lapotre, G. Gogniat, J.-P. Diguet, S. Haddad, A. Baghdadi, **An analytical approach for sizing of heterogeneous multiprocessor flexible platforms for iterative demapping and channel decoding**, in Proc. of 2012 IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2012.

2.7.5 Communications internationales avec comité de lecture sans actes

[MUS 2018d] M. Mushtaq, A. Akram, M. Bhatti, V. Lapotre, G. Gogniat, **Cache-Based Side-Channel Intrusion Detection using Hardware Performance Counters**, 16th International Workshops on Cryptographic architectures embedded in logic devices, Lorient, France, 2018.

[MIG 2015b] V. Migliore, M. Méndez Real, V. Lapotre, G. Gogniat, A. Tisserand, C. Fontaine, **Somewhat homomorphic encryption schemes : which candidates and which expectations to have with this type of encryption schemes ?**, 13th International Workshops on Cryptographic Architectures Embedded in Reconfigurable Devices, 2015.

2.7.6 Communications internationales invitées sans actes

[BON 2017a] G. Bonnoron, C. Fontaine, G. Gogniat, V. Herbert, V. Lapotre, V. Migliore, A. Roux-Langlois, **Somewhat/Fully Homomorphic Encryption : Implementation Progresses and Challenges**, In : El Hajji S., Nitaj A., Souidi E. (eds), Codes, Cryptology and Information Security. C2SI 2017. Lecture Notes in Computer Science, vol 10194. Springer, Cham, 2017.

[LAP 2014a] V. Lapotre, **A Dynamically Reconfigurable Multi-ASIP Architecture for Multi-Standard and Multi-Mode Turbo Decoding**, International Symposium on System-on-Chip, 2014.

2.7.7 Communications nationales avec comité de lecture et actes

[WAH 2017b] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre, G. Gogniat, **ARM-HEx : a hardware extension for information flow tracking on ARM-based platforms**, Les Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI 2017), 2017.

[MEN 2016d] M. Méndez Real, V. Lapotre, G. Gogniat and V. Migliore, **Physical isolation against cache-based Side-Channel Attacks on NoC-based architectures**, Compas'2016.

[MIG 2016b] V. Migliore, M. Méndez Real, V. Lapotre and G. Gogniat, **Algorithmes pour le chiffrement homomorphe**, Compas'2016.

[LAP 2013h] V. Lapotre, P. Murugappa, G. Gogniat, A. Baghdadi, J.-P. Diguet, **Plateforme multi-ASIP reconfigurable dynamiquement pour le turbo décodage dans un contexte multi-standard**, in Proc. of Gretsi, 2013.

[LAP 2011a] V. Lapotre, P. Coussy, and C. Chavet, **Prédiction de branchement dans la synthèse de haut niveau**, in Proc. of SYMPosium en Architectures, Saint Malo, 2011.

2.7.8 Communications nationales invitées sans actes

[LAP 2019a] V. Lapotre, **A Hardware/software co-design approach for security analysis of application behavior - Applications on Dynamic Information Flow Tracking** , Journée "Nouvelles Avancées en Sécurité des Systèmes d'Information", Toulouse, 2019.

[LAP 2018a] V. Lapotre, **Déploiement sécurisé d'applications au sein des architectures many-cœurs**, FETCH 2018.

[LAP 2017b] V. Lapotre, **Sécurité des objets, Workshop InS3pect - Ingénierie Systèmes de Services Sécurisés Pour objEts ConnecTé**, Sophia Antipolis, 2017.

2.7.9 Communications par affiches / Démonstrations (sélection)

[WAH 2017c] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre, G. Gogniat, **ARM-HEx : a framework for efficient DIFT in real-world SoCs**, 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.

[DEV 2015a] C. Dévigne, J-B. Bréjon, Q. Meunier, F. Wajsbürt, M. Mendez Real, V. Migliore, V. Lapotre, G. Gogniat, M. Aichouch, M. Ait Hmid, C. Mancillas López, L. Bossuet, V. Fischer, **Applications security in manycore platform, from operating system to hypervisor : how to build a chain of trust**, CHES 2015 Workshop on Cryptographic Hardware and Embedded Systems, 2015.

[MIG 2015c] V. Migliore, M. Méndez Real, V. Lapotre, G. Gogniat, A. Tisserand, C. Fontaine, **Exploration of the best polynomial multiplication algorithm for homomorphic encryption schemes**, CHES 2015 Workshop on Cryptographic Hardware and Embedded Systems, 2015.

[GAR 2014a] R. Garibotti, L. Ost, A. Gamatié, V Lapotre, C. J. Adeniyi-Jones and G. Sassatelli, **Multithreading for Compute Accelerators Through Distributed Shared Memory Design**, Design Automation Conference (DAC'2014), Work-in-Progress Session, 2014.

2.7.10 logiciel libre

PAnTHErS Outils pour l'évaluation de schémas cryptographiques homomorphes. PAnTHErS fournit une estimation rapide du temps d'exécution et de l'empreinte mémoire pour un jeu de paramètres représentatif du besoin applicatif. PAnTHErS fournit une solution d'exploration permettant à l'utilisateur de choisir un schéma de chiffrement et les paramètres as-

sociés en tenant compte des besoins applicatifs. L'outil est disponible via le lien suivant : `https://github.com/cferon/PAnTHErS`.

3 Activités d'enseignement

J'ai réalisé mes premières missions d'enseignement durant ma thèse de doctorat entre 2010 et 2013. Durant cette période, tous mes enseignements ont été réalisés au sein de l'UFR Sciences et Sciences de l'Ingénieur (SSI) de l'Université Bretagne Sud. Plus précisément au sein de la filière Électronique. J'ai intégré une équipe pédagogique disponible et dynamique qui m'a permis de découvrir l'enseignement et d'en appréhender la fonction au travers de nombreuses discussions. Le bilan très positif de ces années, sur le plan professionnel et personnel, a alors confirmé ma volonté de devenir Maître de Conférences.
Durant cette période, j'ai pu couvrir les vastes domaines de l'électronique : électronique analogique, électronique numérique, traitement du signal, automatisme, et programmation informatique.

Depuis ma nomination en septembre 2014, j'effectue la grande majorité de mes enseignements au sein de la spécialité Cyberdéfense de l'Ecole Nationale Supérieure d'Ingénieurs de Bretagne Sud (ENSIBS). Cette formation, créée en 2013, vise à former des ingénieurs capables de cyberdéfendre les Opérateurs d'Importance Vitale (OIV) français. Cette formation, co-construite avec les entreprises, possède la particularité d'être uniquement proposée par apprentissage. Dans ce cadre, j'ai accompagné la mise en place de cette formation un an après sa création. J'interviens principalement dans les enseignements relevant du domaine de la sécurité des systèmes embarqués, de l'électronique numérique et de la programmation.

Ces dernières années j'ai également enseigné dans le Parcours École d'Ingénieurs (PEI) de l'ENSIBS accessible post-bac ainsi que dans le Master Cyber-Sécurité des Systèmes Embarqués (CSSE) de l'UFR Sciences et Sciences de l'Ingénieur (SSI) de l'Université Bretagne Sud.
Dans la suite je présente chacune de mes expériences d'enseignement depuis mon doctorat. Pour plus de clarté, les volumes d'enseignement ne sont pas précisés pour chaque enseignement mais sont regroupés dans les tables présentées dans la section 3.3 du document.

3.1 De 2010 à 2013 en tant que doctorant contractuel

Dans le cadre de mon contrat doctoral avec enseignement, j'ai effectué 192 heures d'enseignement à l'UFR SSI de l'Université Bretagne Sud. Ces enseignements ont principalement été réalisés en troisième année de Licence au sein du parcours Electronique et Informatique Industrielle (EII) de la licence Physique Chimie Sciences de l'Ingénieur (PCSI).

3.1.1 Licence 3 PCSI, parcours EII (2010/2013)

Automatisme - Encadrement de travaux pratiques. Ces TPs permettent aux étudiants d'appréhender la programmation d'automates, l'analyse d'un cahier des charges et l'apprentissage des langages Grafcet, à contacts et littéral structuré.

Électronique numérique - Encadrement de travaux pratiques. Ces TPs permettent aux étudiants d'appréhender la technologie des composants numériques, d'analyser et concevoir des architectures numériques.

Création partielle de l'enseignement
Électronique analogique - Encadrement de travaux pratiques. Ces TPs permettent aux étudiants d'apprendre à utiliser un oscilloscope, de mettre en oeuvre les filtres passifs du 1er et 2ème ordre, les filtres actifs ainsi que les amplificateurs opérationnels en régime linéaire et non linéaire.

Création partielle de l'enseignement
Programmation informatique - Encadrement de travaux pratiques et de projets en langage C. Ces enseignements mettent en pratique les connaissances acquises en langage C et permettre de

se familiariser avec les bonnes pratiques de développement, l'écriture de Makefile et le debogage d'applications.

3.1.2 Master 1 Math-STIC, parcours SIAM (2013)

Création de l'enseignement

Électronique numérique - Mise en place d'un cours magistral sur les architectures reconfigurables et encadrement de de travaux pratiques permettant aux étudiants de réaliser du prototypage sur cible FPGA.

3.1.3 Licence 1 Sciences de l'Ingénieur (SI) (2013)

Électronique numérique - Encadrement de de travaux pratiques. Ces TPs permettent aux étudiants de mettre en œuvre une chaîne de mesure numérique (capteurs, échantillonnage, et quantification).

3.2 Depuis 2014 en tant que Maître de Conférences

Depuis septembre 2014, je suis en poste à l'école d'ingénieurs ENSIBS au sein l'équipe pédagogique de la spécialité Cyberdéfense. Cette spécialité, accessible uniquement par la voie de l'apprentissage, a recruté sa première promotion en 2013. Mon recrutement a donc accompagné la création de cette spécialité qui accueille aujourd'hui 84 apprentis par promotion. Par conséquent, la création d'enseignements a été au cœur de mes premières années d'enseignant. Certains de ces enseignements sont mutualisés avec la spécialité Cybersécurité du logiciel de l'ENSIBS. De plus, j'ai créé des enseignements pour le parcours post-bac (PEI) de l'ENSIBS et pour le Master Cyber-Sécurité des Systèmes Embarqués créé en 2017 à l'UFR SSI de Lorient.

3.2.1 Diplômes d'ingénieur ENSIBS

Diplôme d'ingénieur en Cyberdéfense

Création de l'enseignement

Sécurité matérielle - 2ème année du cycle ingénieur - depuis 2014

Cet enseignement est une introduction à la sécurité matérielle. Il permet d'appréhender les enjeux liés à la sécurité des composants électroniques et de connaître les principes de base permettant la génération de nombres vraiment aléatoires. Dans un premier temps, un panorama de la menace est réalisé. Celui-ci concerne les attaques physiques menant à l'extraction d'information sensible (retro-conception, attaques par perturbations et observations), les risques dans le cycle de vie des composants (vol de la propriété intellectuelle, contrefaçon, insertion de chevaux de Troie matériels...). Dans un second temps, les principales contremesures sont présentées. Dans un troisième temps, une mise en situation pratique permet l'étude d'attaques par observation sur une cible microcontrôleur. À l'aide d'une sonde électromagnétique et d'un oscilloscope, les apprenants réalisent des campagnes de mesures puis analysent les traces obtenues pour en déduire de l'information sur les traitements réalisés (comportements des branchements, variations temporelles des traitements en fonction des données manipulées, corrélation entre les valeurs manipulées et le rayonnement électromagnétique). Enfin, une quatrième partie de cet enseignement présente les bases de la conception de générateurs de nombres vraiment aléatoires.

Diplôme d'ingénieur en Cyberdéfense et Cybersécurité du logiciel
Création de l'enseignement

Microprocesseur et langage d'assemblage - 1ère année du cycle ingénieur - depuis 2014

Les objectifs de cet enseignement sont de connaître les principes de base de la conception de circuits numériques, l'organisation et le fonctionnement d'un ordinateur, les principes de fonctionnement d'un microprocesseur, d'appréhender les enjeux de sécurité au sein des microprocesseurs et de connaître et maîtriser les bases de la programmation en langage d'assemblage sur cible x86. Cet enseignement se divise en deux parties. La première se concentre sur le fonctionnement et l'architecture des microprocesseurs (hiérarchie mémoire, unités fonctionnelles, pipeline, exécution dans le désordre...). Une mise en situation pratique permet la construction et la simulation d'un processeur simple (construction du chemin de données, définition du jeux d'instructions...). La seconde partie se concentre sur l'apprentissage du langage d'assemblage pour les architectures x86. Pour mettre en pratique les connaissances acquises, un projet de développement en langage d'assemblage est proposé.

Diplôme d'ingénieur en Cyberdéfense et Cybersécurité du logiciel
Création de l'enseignement

Linux embarqué - 2ème année du cycle ingénieur - 2014-2021

Cet enseignement a pour objectif de connaître les spécificités des systèmes embarqués, connaître l'organisation d'un système Linux, construire une distribution Linux personnalisée pour les systèmes embarqués et appréhender les enjeux de cybersécurité au sein des systèmes embarqués. Dans ce module, après un cours magistral permettant de transmettre les éléments de connaissance minimale à la mise en pratique, un projet est au cœur de l'acquisition des compétences. Ce projet s'appuie sur un mini-ordinateur *Raspberry* et l'outil buildroot pour la création d'une distribution Linux sur mesure. En fin de projet, l'accent est particulièrement mis sur la sécurisation du système mis en oeuvre (authentification, droits d'accès, pare-feu, moindre privilège, journaux d'événements...).

Diplôme d'ingénieur en Cyberdéfense
Création de l'enseignement

Programmation en langage Python - 1ère année du cycle ingénieur - 2014-2015

Le langage de programmation python est grandement utilisé dans les domaines de la cybersécurité dans le but de développer des scripts d'automatisation de tâches, d'effectuer de l'analyse de données et de concevoir des plugins d'outils. Cet enseignement permet aux apprenants de se familiariser avec ce langage. Après une présentation des éléments clés du langage, cet enseignement se compose principalement de travaux pratiques permettant l'apprentissage de la syntaxe du langage, des structures de données, de l'utilisation de library de calcul (p.ex. *numpy*) ainsi que des bonnes pratiques de codage (règles de codage, utilisation de l'interpréteur, vérification de types, pièges du langage...).

Diplôme d'ingénieur en Cyberdéfense
Création de l'enseignement

Programmation en langage C - 1ère année du cycle ingénieur - 2014-2018

Le langage est au cœur de nombreux systèmes (en particulier les systèmes d'exploitation). Maîtriser ce langage est indispensable pour de nombreux aspects de la cybersécurité (audit de code, recherche de vulnérabilités, retro-conception de logiciels malveillants...). Cet enseignement a pour objectif de permettre à l'apprenant de maîtriser les concepts clés du langage C, de se familiariser avec les outils de débogage, de mettre en oeuvre des règles de programmation (et de compilation) pour le développement sécurisé. Après une présentation des éléments clés du langage, cet enseignement se compose principalement de travaux pratiques permettant de se familiariser

avec le langage (types, structures, pointeurs...) et d'appréhender les liens avec les problèmes de sécurité (débordement de tampons/pile, manipulations illicites de la mémoire, détournement du flot d'exécution...).

Diplôme d'ingénieur en Cyberdéfense
Création de l'enseignement
Programmation antireverse - 2ème année du cycle ingénieur - 2014-2018
Cet enseignement est une introduction aux techniques dites "antireverse" permettant de fortement complexifier la tâche d'un individu malveillant cherchant à rétro-concevoir un logiciel pour en extraire la propriété intellectuelle (il est important de noter que ces approches sont également utilisées par des concepteurs de logiciels malveillants afin de complexifier la travail des analystes). Cet enseignement repose entièrement sur une pédagogie inductive. Durant chaque séance, les apprenants étudient et mettent en oeuvre un ensemble de techniques (obscurcissement, chiffrement, détection de débogage...). La complexité, l'efficacité, les limites de chaque approche sont alors discutées par l'ensemble du groupe apprenants.

3.2.2 Master Cyber-Sécurité des Systèmes Embarqués

Création partielle de l'enseignement
Programmation défensive - 1ère année de Master - 2017-2019
Cet enseignement se focalise sur les règles de programmation (et de compilation) pour le développement sécurisé en langage C pour un public ayant une bonne maîtrise de ce langage. La partie cours magistral de cet enseignement présente les erreurs communes de développement pouvant mener à l'intégration de vulnérabilités dans un logiciel, les options de compilations permettant de rendre plus robuste le binaire généré, les outils d'analyse statique et dynamique pour la sécurité et les cycles de développement intégrant la composante cybersécurité. Les travaux dirigés permettent aux apprenants de mettre en pratique les notions vues en cours en s'appuyant sur le compilateur GCC. Les apprenants sont amenés à analyser des codes vulnérables et à proposer des corrections. Enfin, un projet de développement est réalisé dans le cadre de travaux pratiques (p.ex. un gestionnaire de mot de passe sous Linux).

Création de l'enseignement
Sécurité de la microarchitecture - 2ème année de Master - depuis 2017
Cet enseignement a pour objectif de présenter les vulnérabilités des architectures de processeurs modernes. Une première partie sous la forme d'un cours magistral, présente les enjeux de sécurité en s'appuyant sur des attaques comme Meltdown, Spectre ou des attaques exploitant des canaux axillaires via des ressources partagées (mémoires caches, unités fonctionnelles, prédicteurs de branchement...). Dans une seconde partie, les apprenants mettent en oeuvre, en travaux dirigés, une preuve de concept d'analyse par canaux auxiliaire s'appuyant sur les mémoires caches ciblant une implémentation vulnérable d'un algorithme cryptographique. Enfin, durant les travaux pratiques, les apprenants s'appuient sur les connaissances acquises durant l'analyse de la preuve de concept pour mettre en place un canal de communication caché entre deux processus en s'appuyant sur les mémoires caches.

Création de l'enseignement
Sécurité matérielle - 1ère année de Master - depuis 2021
Cet enseignement est une introduction à la sécurité matérielle. Il permet d'appréhender les enjeux liés à la sécurité des composants électroniques. Dans un premier temps, un panorama de la menace est réalisé. Celui-ci concerne les attaques physiques menant à l'extraction d'information sensible (retro-conception, attaques par perturbations et observations), les risques dans le cycle de vie des composants (vol de la propriété intellectuelle, contrefaçon, insertion de chevaux de Troie matériels...). Dans un second temps, les principales contremesures sont présentées. Dans

un troisième temps, une mise en situation pratique permet l'étude d'attaques par observation sur une cible microcontrôleur. À l'aide d'une sonde électromagnétique et d'un oscilloscope, les apprenants réalisent des campagnes de mesures puis analysent les traces obtenues pour en déduire de l'information sur les traitements réalisés (comportements des branchements, variations temporelles des traitements en fonction des données manipulées, corrélation entre les valeurs manipulées et le rayonnement électromagnétique).

3.2.3 PEI ENSIBS

Création de l'enseignement

Réseaux informatiques - 2ème année de PEI - 2015-2020

Les réseaux informatiques sont aux cœurs de toutes les spécialités de l'ENSIBS. C'est pourquoi, en 2015, nous avons créé une unité d'enseignement complète sur cette thématique à destination des apprenants de notre classe préparatoire intégrée. Durant cet enseignement, une première partie sous la forme de cours magistraux permet d'introduire le vocabulaire et les concepts clés des réseaux informatiques (topologies, modèles en couches, encapsulation, protocoles...). Dans une seconde partie, un volume horaire important de travaux pratiques encadrés est réalisé. Une première étape permet de mettre en place un réseau local et d'en analyser le fonctionnement (DHCP, DNS, analyse de trames). Une seconde étape permet de se familiariser avec les réseaux de terrains (i2c, CAN...). Une troisième étape permet de découvrir la programmation réseau (UDP, TCP) en langage python. Enfin, un projet de groupe est réalisé en fin d'enseignement afin de construire collectivement un petit système industriel configurable et pilotable à distance.

3.3 Résumé de l'activité d'enseignement depuis 2010

3.3.1 de 2010 à 2013 (Doctorant contractuel)

La Table 4 présente la répartition des enseignements dont j'ai eu la responsabilité durant mes trois années de doctorat au sein de l'UFR Sciences et Sciences de l'Ingénieur (SSI) de l'Université Bretagne Sud. Le volume total représente 192 heures quasi uniquement sous la forme de travaux pratiques.

Table 4 – Résumé des enseignements réalisés durant mon doctorat

	niveau	10/11	11/12	12/13
Licence 1 Sciences de l'Ingénieur				
Électronique numérique	Bac+3			16h TP
Licence 3 PCSI, parcours EII				
Électronique numérique	Bac+3	8h TP		
Électronique analogique	Bac+3	10h CM - 6h TD - 20h TP		
Programmation informatique	Bac+3	12h TP		
Automatisme	Bac+3			16h TP
Master 1 Math-STIC, parcours SIAM				
Électronique numérique	Bac+3			6h CM -22h TP

3.3.2 de 2014 à aujourd'hui

La Table 5 présente la répartition des enseignements dont j'ai eu la responsabilité depuis 2014 au sein de l'école d'ingénieurs ENSIBS et de l'UFR Sciences et Sciences de l'Ingénieur (SSI) de l'Université Bretagne Sud.

La Table 5 montre que mon service d'enseignement s'est, au fil des ans, recentré sur les thématiques proches de mes activités de recherche : la sécurité des architectures matérielles. Cela

TABLE 5 – Résumé des enseignements réalisés depuis 2014

	niveau	14/15	15/16	16/17	17/18	18/19	19/20	20/21	21/22
ENSIBS cycle ingénieur									
Sécurité matérielle	Bac+4	9h CM - 2h TD - 12h TP							
Microprocesseur et langage d'assemblage	Bac+3	10h CM - 6h TD - 20h TP							
Linux embarqué	Bac+4	4h CM - 10h TP							
Programmation en langage Python	Bac+3	2h CM - 16h TP							
Programmation en langage C	Bac+3	2h CM - 20 h TP							
Programmation antireverse	Bac+4	10h TD							
ENSIBS cycle PEI									
Réseaux informatiques	Bac+2	4h CM - 20h TP							
UFR SSI Master CSSE									
Programmation défensive	Bac+4				2h CM - 4h TD - 12h TP				
Sécurité de la microarchitecture	Bac+5				2h CM - 6h TD - 6h TP				
Sécurité matérielle	Bac+4								4h CM 4h TD 8h TP

a été rendu possible par la montée en puissance de la filière Cyberdéfense de l'ENSIBS qui est passée de promotions de 25 apprentis en 2014 (1 groupe TD - 2 groupes TP) à des promotions de 84 apprentis à partir de 2020 (3 groupes TD - 6 groupes TP). De plus, à partir de 2020, la filière Cybersécurité du logiciel a également augmenté ses effectifs en passant de 1 groupe TD à 2 groupes TD. De plus, depuis 2018, l'obtention d'une PEDR a limité le nombre d'heures complémentaires pouvant être réalisées à 96h équivalent TD en cas d'activités au sein d'une filière par apprentissage.

Mes activités d'enseignement au sein de la spécialités Cyberdéfense de l'ENSIBS sont complétées par des tâches d'encadrement de projets (3 par an en moyenne) et par le tutorat d'apprentis tout au long de leur parcours de formation (6 par an en moyenne). De plus, j'encadre également un projet par an au sein du parcours post bac de l'ENSIBS (PEI) et au sein du Master CSSE de l'UFR SSI.

3.4 Bilan et réflexion sur la fonction d'enseignant

Je réalise des enseignements au sein de l'Université Bretagne Sud depuis maintenant plus de dix ans. L'écriture de ce manuscrit est également pour moi l'opportunité de mener une réflexion sur le métier d'enseignant du supérieur. Dans la suite, je partagerai quelques éléments de réflexion qui ont marqué mes expériences depuis 2014.

Si le rôle premier de l'enseignant est de transmettre des connaissances, les leviers pédagogiques pour atteindre ce but sont multiples. Le choix d'un levier plutôt qu'un autre n'est pas trivial et s'appuie sur de nombreux critères. Ce choix peut s'appuyer, par exemple, sur des éléments tels que : la nature du public cible (début / fin du cycle universitaire, statut apprentis / étudiants), le type de compétences visées (théoriques, pratiques, méthodologiques), le volume horaire alloué (encadré et non encadré), la place de l'enseignement au sein de la formation (cœur de métier, socle pour l'acquisition de compétences futures, ouverture culturelle et scientifique),... Ces éléments, très *scolaires*, ne tiennent pas toujours compte des spécificités sociétales et générationnelles. Par exemple, l'accès rapide à une masse d'information via internet a grandement changer les reflexes des apprenants face à une difficulté. J'ai pu observer ces dernières années que les élèves ingénieurs favorisent majoritairement une recherche sur internet plutôt que dans les ressources

fournies par l'enseignant. Ce réflexe peut parfois s'avérer préjudiciable en raison du temps de traitement des informations (lecture de tutoriaux, forums, vidéos, ...) et de la qualité variable des ressources disponibles en ligne. L'outil Internet possède, à mon avis, toute sa place dans les enseignements. Cependant, il est nécessaire de prendre le temps d'accompagner les apprenants dans son utilisation. Cela peut passer par la définition collective de mots clés, l'apprentissage des méthodes de recherche avancées, la présélection par l'enseignant de ressources disponibles en ligne ou encore la confrontation en groupe des sources d'information.

Le point que nous venons d'aborder est naturellement lié à celui concernant un phénomène relativement récent : l'hyperconnectivité. Nous vivons dans une société où chaque individu possède un smartphone connecté en permanence au réseau Internet. Les apprenants que nous accueillons aujourd'hui ont grandi, parfois depuis le collège, en utilisant quotidiennement un smartphone pour s'informer, communiquer ou se divertir. Il est alors naturel pour eux d'interagir très régulièrement (plusieurs fois par heure) avec leur smartphone. Ce phénomène, qui détourne souvent l'attention des apprenants, doit aujourd'hui être pris en compte par l'enseignant lors de la préparation de son enseignement. Évidemment, la solution *zero smartphone/connexion* pendant les séances peut toujours être appliquée. Cette solution, réaliste sur un temps d'enseignement court, par exemple un cours magistral d'une heure, ne me semble pas viable sur des créneaux plus long de travaux pratiques par exemple. Je pense qu'il est nécessaire de transformer ce phénomène en opportunité pour dynamiser nos enseignements. Nous voyons déjà apparaître des solutions permettant aux enseignants d'interagir en temps réel avec les apprenants via des applications sur smartphone ou accessibles via un navigateur web. Ces solutions permettent de faire entrer les habitudes numériques des apprenants dans les salles de cours via des questionnaires, des sondages ou des commentaires qui permettent d'augmenter significativement l'interaction entre enseignants et apprenants. La pandémie de COVID-19 a été un accélérateur pour ce type de solutions dans le cadre d'un enseignement à distance. Néanmoins, il est maintenant nécessaire de gagner en maturité vis à vis des solutions numériques pour une pédagogie innovante dans le but de trouver l'équilibre entre les échanges oraux et les échanges numériques.

Dans ce dernier paragraphe, j'aborderai succinctement deux éléments de réflexions supplémentaires qui me semblent également importants. Durant mes premières années d'enseignement suite à mon recrutement en 2014, je réalisais beaucoup plus d'heures de cours magistraux qu'aujourd'hui. Ce format a l'avantage de permettre la transmission d'une quantité importante d'information en un temps relativement court. Cependant, ce format atteint ses limites quand il devient, au yeux de son public, un moyen dépassé de transmettre des connaissances. Nous avons vu dans le paragraphe précédent que des outils numériques peuvent aider à dynamiser ce format. Ces dernières années, j'ai choisi d'expérimenter une démarche *inductive* qui réduit la place des cours magistraux dans le volume horaire alloué à mes enseignements. Le volume horaire libéré est alors utilisé pour réaliser des travaux pratiques plus ambitieux, par exemple via des mini projets, qui permettent d'aborder des notions théoriques et techniques avancées au fur et à mesure de l'apprentissage. Je réalise alors "à la demande" des *vignettes* courtes assimilables à des micro cours magistraux pendant les séances de travaux dirigés et pratiques. L'accueil de cette approche par les apprenants est aujourd'hui positive car contrairement à l'approche *déductive*, les connaissances nécessaires à la résolution d'un problème sont apportées lorsque que le besoin est réel. Cela permet d'ancrer plus facilement la notion dans le concret. Cette volonté d'expérimenter la démarche inductive m'a amené à travailler sur la notion d'amélioration continue dans le cadre de mes enseignements. J'ai choisi d'aborder ce travail par trois axes principaux : les apprenants, la veille technologique et la recherche. Tout d'abord, les apprenants sont sollicités à chaque fin d'enseignement via un questionnaire anonyme me permettant de relever les forces et les faiblesses selon le public cible. La veille technologique me permet de faire évoluer mes enseignements régulièrement via l'utilisation de nouvelles plateformes ou outils. Par exemple, la montée en puissance du jeu d'instructions libre RISC-V est une opportunité intéressante dans le cadre de l'enseignement Microprocesseur et langage d'assemblage dont je suis responsable. Enfin,

il est important pour moi de connecter mes enseignements aux travaux de recherche quand cela est possible. Cela passe, par exemple, par l'évocation de travaux de recherche de la communauté durant mes enseignements afin d'éveiller la curiosité des apprenants. Cela permet bien souvent d'engager des discussions concernant le monde de la recherche et les opportunités de poursuites en thèse de doctorat.

Le métier d'enseignant du supérieur est un métier extrêmement riche. L'accompagnement des apprenants vers un métier épanouissant et de haut niveau nécessite une réflexion permanente sur l'évolution des métiers, des technologies et de la société. Les pistes pour améliorer nos enseignements sont nombreuses et méritent que nous les explorions afin de répondre au mieux aux besoins de notre société. De plus, en tant qu'enseignant-chercheur, notre rôle est également d'éveiller la curiosité scientifique des apprenants qui seront également les chercheurs de demain.

4 Responsabilités collectives, animations et projets scientifiques

4.1 Au niveau de l'Université de Bretagne Sud

4.1.1 Au sein de l'ENSIBS

Depuis 2015, je suis en charge d'une fonction transverse au sein de l'ENSIBS : responsable du recrutement. Différentes missions sont associées à cette responsabilité.

La première concerne l'organisation du recrutement de nos étudiants sous statut étudiant. Cela passe par la définition, la rédaction et la mise en oeuvre de la procédure de recrutement. J'assure alors la coordination des personnels administratifs recevant et vérifiant les dossiers de candidature, j'organise les jurys d'étude des dossiers de candidature ainsi que les entretiens de motivation qui mobilisent tous les collègues enseignants de l'école.

La seconde mission concerne la gestion des concours à bac+2. L'ENSIBS recrute des étudiants issus de CPGE MP, PC, PSI, TSI, et PT via 3 concours (e3a Polytech, Banque PT, CCINP). Mon rôle est alors de gérer cette voie de recrutement (jurys d'admissions et appels) qui se déroule de juin à septembre et d'assurer le lien entre l'ENSIBS et les organisateurs des concours (présence dans les jurys d'admissions, les assemblés générales, et les réunions occasionnelles).

La troisième mission concerne le maintien à jour d'une base de données concernant le recrutement (origine géographique, établissement d'origine, nombres de candidatures, taux de féminisation...). Ces informations permettent par exemple d'orienter les moyens de communication de l'école (forums, salons) ou d'analyser le taux de pression sur nos filières (nombre de candidats ou nombre d'admis rapporté au nombre final d'inscrits).

Entre 2015 et 2018, j'ai également été référent pédagogique de la première année du cycle ingénieur en Cyberdéfense. Parmi les tâches associées à cette responsabilité, on trouve : l'accueil des étudiants à la rentrée, les bilans de semestre, l'organisation et l'animation des jurys de semestre, la réalisation du lien entre les apprentis et la direction des études.

Entre 2014 et 2018, j'ai représenté l'ENSIBS, avec le directeur de l'école, au club formation du Pôle d'Excellence Cyber (PEC).

4.1.2 Au sein du laboratoire Lab-STICC

Entre 2015 et 2019, j'ai été correspondant pour l'équipe MOCS au sein de l'axe de recherche transverse Cyrus (Cybersécurité) du laboratoire Lab-STICC.

Pendant 2 ans (2020-2021), j'ai animé le "groupe de travail RISC-V" au sein de l'équipe ARCAD du laboratoire Lab-STICC. Ce groupe rassemble les membres de l'équipe s'intéressant aux processeurs basés sur le jeu d'instructions libre RISC-V. Il permet d'échanger sur les activités

de recherche de ses membres (présentations et démonstrations) et de présenter des travaux de l'état de l'art.

Depuis septembre 2021, je suis responsable de l'équipe de recherche ARCAD (`https://labsticc.fr/fr/equipes/arcad`), équipe composée de 14 chercheurs permanents répartis sur 4 établissements (UBS, UBO, IMTA, ENSTA-Bretagne). Cette mission regroupe des tâches liées à l'animation scientifique, la représentation des membres de l'équipes dans les instances, la communication et la vie du laboratoire. De plus, depuis avril 2022, je suis membre élu du conseil de laboratoire du Lab-STICC.

4.1.3 Au sein des conseils d'établissement

Durant mon doctorat, j'ai choisi de participer à la vie universitaire de l'Université de Bretagne Sud. En accord avec cette volonté et ma position de doctorant, j'ai été élu au conseil scientifique de l'Université pour un mandat de deux ans (2012-2013).

4.2 Au niveau national

Depuis ma nomination je me suis toujours attaché à participer aux réseaux scientifiques. Je suis membre des groupements de recherche SOC2 (System On Chip - Systèmes embarqués et Objets Connectés), sécurité informatique et IM (Informatique Mathématique).

En 2019, j'ai organisé avec Alberto Bosio la journée thématique du GDR SOC2 "Sécurité, fiabilité et test des SoC2 : challenges et opportunités dans l'ère de l'Intelligence Artificielle".

4.3 Participation à des collaborations scientifiques et à des contrats d'études

4.3.1 Collaborations Académiques Internationales

Information Technology University (ITU), Lahore,Pakistan

SECUREORDO
Type : bourse Université Bretagne Sud
Durée : 2015
Partenaires : Guy Gogniat (Lorient), Vianney Lapôtre (Lorient), Khurram BHATTI (Lahore)
Objet : Prise en compte de la sécurité dans les méthodes d'ordonnancement
Contribution du Lab-STICC : Impact de l'ordonnancement sur les attaques par canaux auxiliaires s'appuyant sur les mémoires caches
Budget : 4k€(Lab-STICC) - Accueil de Khurram BHATTI pendant 2 mois

e-health.SECURE
Type : PHC PERIDOT
Durée : 2017 - 2019
Partenaires : Guy Gogniat (Lorient), Vianney Lapôtre (Lorient), Khurram BHATTI (Lahore)
Objet : Sécurité des systèmes numériques pour la santé
Contribution du Lab-STICC : définition d'une technique permettant de détecter les attaques par canaux auxiliaires s'appuyant sur les mémoires caches
Budget : 10k€(Lab-STICC) - Séjour d'une doctorante (3 mois) et d'un enseignant chercheur (3 semaines) à l'ITU

PROTECTCACHE
Type : bourses d'excellence Eiffel - Volet Doctorat
Durée : 2017 - 2018
Partenaires : Guy Gogniat (Lorient), Vianney Lapôtre (Lorient), Khurram BHATTI (Lahore)
Objet : Architecture de caches robuste aux attaques par canaux auxiliaires s'appuyant sur les

mémoires caches
Contribution du Lab-STICC : Définition d'une architecture de cache avec aléa
Budget : 12k€- Séjour d'un doctorant (10 mois) à l'UBS

Ruhr-Universität Bochum, Allemagne

SAFEPARA
Type : Bourse Université Bretagne
Durée : 2015
Objet : Simulation d'architecture many-core avec OVPSim
Partenaires : Guy Gogniat (Lorient), Vianney Lapôtre (Lorient), Diana GOEHRINGER (Bochum)
Contribution du Lab-STICC : Définition d'un modèle d'architecture manycore pour OVPSim
Budget : 3k€(Lab-STICC) Accueil de Diana GOEHRINGER pendant 1 mois

MULTISEC
Type : Bourse Université Bretagne Sud et Ruhr-Universität Bochum
Durée : 2015
Objet : Simulation de système manycore sous contrainte de sécurité
Partenaires : Guy Gogniat (Lorient), Vianney Lapôtre (Lorient), Diana GOEHRINGER (Bochum)
Contribution du Lab-STICC : Mise en oeuvre de zone de sécurité dans un modèle d'architecture manycore
Budget : 6k€(Lab-STICC) Séjour de 4 mois d'une doctorante à l'université de Bochum

University of Massachusetts, USA

CRYPHW
Type : Bourse Université Bretagne Sud et University of Massachusetts
Durée : 2016
Objet : Accélération matérielle sur FPGA pour la cryptographie homomorphe
Partenaires : Guy Gogniat (Lorient), Vianney Lapôtre (Lorient), Russell Tessier (Amherst)
Contribution du Lab-STICC : définition d'un accélérateur matériel pour la multiplication via l'algorithme de Karatsuba
Budget : 7k€(Lab-STICC) Séjour de 3 mois d'un doctorant à l'University of Massachusetts

4.3.2 Contrats de Recherche

En tant que responsable scientifique

Securité RISC-V
Type : Micro-projet laboratoire commun LATERAL
Durée : 2022
Objet : Évaluer la sensibilité d'un cœur de processeur RISC-V (softcore sur FPGA) disponible sur étagère face à des attaques par perturbation
Partenaires : Lab-STICC, Thales
Contribution du Lab-STICC : Mise en oeuvre d'un Softcore RISC-V sur cible FPGA et réalisation de campagnes d'injection de fautes
Budget : 6k€(Lab-STICC) - Financement d'un stage

SCRATCHS - Side-Channel Resistant Applications Through Co-designed Hardware/Software
Type : LabEX CominLabs
Durée : 2021 - 2024
Objet : Conception d'un couple processeur-compilateur robuste face à des attaques basées sur l'analyse du temps d'exécution
Partenaires : Lab-STICC, IRISA
Contribution du Lab-STICC : Étude et développement d'une architecture de processeur basé sur le jeu d'instructions RISC-V intégrant des mécanismes de protections face à des attaques basées sur l'analyse du temps d'exécution
Budget : 118k€(Lab-STICC) - Financement de thèse

Usine du futur
Type : CPER
Durée : 2021 - 2027
Partenaires : Lab-STICC, IRDL
Contribution du Lab-STICC, équipe ARCAD, site de l'UBS : Étude et développement d'une plateforme d'évaluation de la sécurité des systèmes embarqués
Budget Lab-STICC équipe ARCAD : 100k€- Financement d'équipements

En tant que co-responsable scientifique

RISC-V
Type : Micro-projet GIS Cormorant
Durée : 2022
Objet : Evaluation des performances d'un cœur RISC-V sur cible FPGA
Partenaires : Lab-STICC, Thales
Contribution du Lab-STICC : Étude, développement et évaluation d'un SoC minimaliste intégrant un processeur RISC-V sur cible FPGA Xilinx
Budget : 8k€(Lab-STICC) - Financement d'un stage

CyberSSI
Type : CPER
Durée : 2015 - 2020
Objet : Centre de recherche et d'expertise en cybersécurité
Partenaires : Lab-STICC, IRISA
Contribution du Lab-STICC : Développement de la plateforme A2C2P2 : Analyse et Attaques par canaux cachés et Perturbations Physiques
Budget Lab-STICC Lorient : 500k€- Financement d'équipements

INS3PECT - Ingénierie Systèmes de Services Sécurisés pour objets connectés
Type : CNRS PEPS
Durée : 2017
Objet : Identifier les verrous technologiques et scientifiques concernant l'ingénierie Systèmes de services sécurisés pour objets connectés et de cartographier les acteurs et compétences au niveau national depuis la conception des objets et de leurs services jusqu'à leur déploiement et mise en œuvre dans des infrastructures dynamiques, largement distribuées.
Partenaires : Lab-STICC, I3S, LEAT, LIG
Contribution du Lab-STICC : Expertise dans le domaine de la sécurité des systèmes embarqués
Budget : 12k€(Lab-STICC) - Financement de missions

HomCrypt
Type : CNRS PEPS SISC INS2I
Durée : 2016 - 2017
Objet : Chiffrement homomorphe logiciel-matériel appliqué au domaine de la santé
Partenaires : Lab-STICC
Contribution du Lab-STICC : Développement d'un accélérateur logiciel-matériel pour le calcul homomorphe
Budget : 30k€(Lab-STICC) - Financement d'un ingénieur

En tant que collaborateur

HardBlare
Type : LabEX CominLabs
Durée : 2015 - 2019
Objet : Conception d'un système de DIFT (Dynamic Information Flow Tracking) pour systèmes embarqués à base de processeurs ARM
Partenaires : Lab-STICC, IRISA
Contribution du Lab-STICC : Étude et développement d'un co-processeur pour le DIFT
Budget : 100k€(Lab-STICC) - Financement d'un Post-doc

PPI Cyberdéfense
Type : Fond d'Amorçage Recherche UBS
Durée : 2015 - 2017
Objet : Structurer la recherche en Cybersécurité à l'UBS
Partenaires : Lab-STICC, IRISA, CRPCC, IREA, LMBA
Contribution du Lab-STICC : Définition d'une méthode pour l'apprentissage de la gestion de crise Cyber ; Sécurité des systèmes industriels
Budget : 270k€(Lab-STICC) - Financement de Post-docs

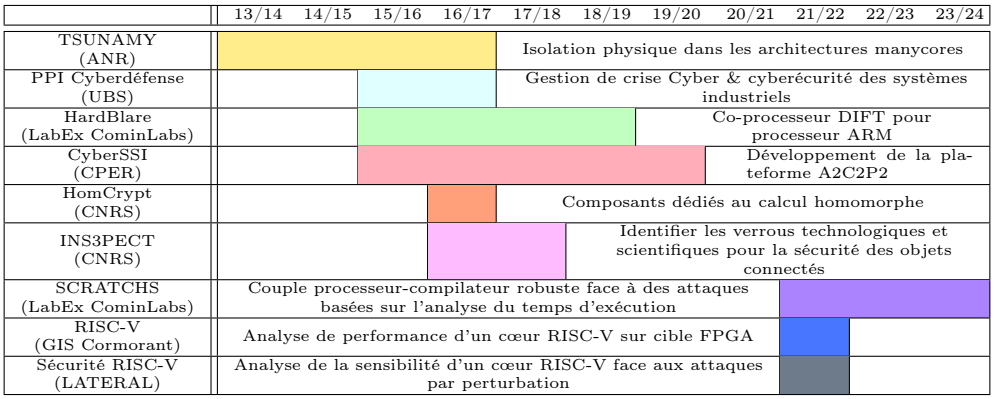
TSUNAMY
Type : ANR
Durée : 2013 - 2017
Objet : Gestion logicielle et matérielle de la sécurité des données pour des plateformes manycore
Partenaires : Lab-STICC, CEA LIST, LIP6, LabHC
Contribution du Lab-STICC : Étude et développement de méthode pour l'isolation physique contre les attaques logiques par canaux auxiliaires basées sur les mémoires caches dans des architectures many-core
Budget : 160k€(Lab-STICC) - Financement de thèse

La Table 6 résume l'ensemble des projets auxquels j'ai participé. Différentes sources de financement ont été utilisées afin de financer les activités de recherche (ANR, LaBex CominLabs, CPER, CNRS, UBS).

4.4 Bilan et réflexion sur la fonction de chercheur

L'obtention d'un poste de Maître de Conférences acte le début d'une carrière d'enseignant-chercheur. Le jeune Maître de Conférences, généralement accompagné par un ou plusieurs chercheurs expérimentés, commence son apprentissage du métier et se prépare pour devenir un chercheur aguerri et autonome. Cette maturité s'obtient au fil du temps via un ensemble de facteurs dont certains me semblent particulièrement importants : l'accumulation des expériences, la connaissance de l'environnement de recherche local, national et international, la réflexion sur le rôle du chercheur, la capacité à se remettre en question et enfin la capacité à se projeter sur le

TABLE 6 – Résumé des projets de recherches hors CPER 2021-2027



court, moyen et long terme.

Depuis 2014, j'ai eu l'occasion de co-encadrer des doctorants, des étudiants de Master, un ingénieur et deux post-doctorants ; j'ai participé à la recherche de financement via le montage de projets de recherche collaboratifs, le dépôt de projets de thèses et de projets de recherche post-doctoral ; j'ai participé à l'organisation d'événements nationaux et internationaux ; j'ai effectué de nombreuses expertises d'articles scientifiques et j'ai réalisé des expertises scientifiques dans le cadre d'appels à projets nationaux ; j'ai participé et me suis impliqué dans la vie de mon laboratoire et j'ai initié un nouveau thème de recherche au sein de mon équipe de recherche. Ces différentes activités m'ont permis de découvrir dans un premier temps, puis, d'approfondir l'éventail des compétences que le métier de chercheur nécessite. Dans la suite de cette discussion, j'aborderai quelques uns des éléments de réflexion sur notre métier qui me semblent importants pour la suite de ma carrière de chercheur.

Les laboratoires dans lesquels nous réalisons nos travaux de recherche baignent dans des environnements socio-économiques riches et complexes. Dans ce contexte, les chercheurs sont amenés, à travers leurs travaux de recherche et leur engagement dans la vie collective, à adresser des problématiques dont le périmètre et les enjeux peuvent être locaux, régionaux, nationaux et/ou internationaux. Une constante demeure toutefois : la recherche a besoin d'échanges pour avancer. Que cela soit au niveau local ou international, partager et confronter les idées et les réflexions est indispensable pour progresser collectivement et individuellement. De plus, pour adresser des défis scientifiques toujours plus complexes, des communautés scientifiques se sont rapprochées afin de construire des approches interdisciplinaires originales. Fort de ces observations, il me semble aujourd'hui indispensable, pour un chercheur, d'être pleinement conscient de l'environnement socio-économique et des spécificités du territoire où il mène ses travaux de recherche tout en construisant des collaborations internationales et interdisciplinaires afin de mener des travaux originaux de haut niveau.

Ces dernières années ont vu s'accroître les difficultés concernant le recrutement d'étudiants souhaitant réaliser des travaux de thèse. S'il est complexe de réaliser une liste exhaustive des raisons ayant menées à cette situation, il est toutefois possible d'identifier quelques éléments d'explication. Tout d'abord, il est évident que la thèse est mal connue des étudiants. Ces derniers ont souvent une image déformée de ce qu'est un travail de thèse. Il est alors indispensable d'organiser des moments d'échanges avec les étudiants de niveau master pour présenter la thèse et ses débouchés et leur permettre de rencontrer des doctorants. De plus, le stage en laboratoire peut permettre de susciter un intérêt pour la recherche académique. Un second facteur important concerne la rémunération des doctorants. Même si cette dernière varie en fonction du type de support, la rémunération d'un doctorant est, dans la majorité des cas, bien plus faible que celle proposée lors d'une première embauche en entreprise après 5 ans d'études. Récemment,

des revalorisations ont permis d'améliorer la situation. Cependant, pour des secteurs en tension, cela ne sera peut être pas suffisant. Par conséquent, le travail de valorisation des compétences qu'apporte le doctorat dans le cadre d'une carrière professionnelle doit continuer. Cet effort de valorisation du doctorat a principalement été mené auprès du monde de l'industrie ces dernières années. Je pense que cet effort doit être renforcé via des actions à l'intention du grand publique et des jeunes étudiants. En effet, si le rôle du chercheur permanent est largement identifié par nos concitoyens, celui du doctorant est peu connu. C'est pourtant une étape clé dans la vie professionnelle d'un chercheur.

Le dernier point que je souhaite aborder dans cette section concerne l'animation scientifique, élément crucial de la vie quotidienne des chercheurs. Quelle que soit sa forme, en initiant ou renforçant les échanges, elle fait progresser les communautés scientifiques. Il est donc important pour un chercheur de s'impliquer dans sa mise oeuvre mais également de s'interroger régulièrement sur les moyens permettant d'en accroitre l'efficacité. Ces dernières années, je me suis particulièrement intéressé à l'animation de groupes de travail locaux. C'est à dire au sein d'un laboratoire ou d'une équipe de recherche. J'ai participé à la mise en place de deux groupes de travail dont un pour lequel j'ai assuré l'animation. Ces groupes réunissent l'ensemble des membres permanents et non-permanents de l'équipe de recherche ARCAD portant un intérêt pour la thématique traitée par le groupe. Ces groupes se réunissent régulièrement afin d'échanger sous différentes formes : présentations de travaux en cours, présentation de travaux issus de l'état de l'art, présentation d'outils, réalisation de démonstrations, partage d'actualités, etc. La mise en place de ces groupes a permis de favoriser les échanges entre les étudiants en thèse ou en projets, les post-doctorants, les ingénieurs, et les chercheurs. Ces échanges, très positifs, permettent notamment d'appréhender différents thèmes de recherche, d'accroître sa culture scientifique, d'échanger sur des points de difficultés et d'ouvrir de nouvelle pistes de réflexions. Ces groupes de travail sont donc, de mon point de vue, un excellent complément aux échanges existants (réunions encadrants-encadrés, séminaires d'équipes, ...).

Je conclurais cette réflexion en parlant du métier de chercheur au pluriel. En effet, il serait, de mon point de vue, plus juste de parler des métiers du chercheur au regards des diverses missions qui lui seront confiées tout au long de sa carrière. Cependant, si je devais associer un seul mot à ce métier, ce serait le verbe *accompagner*. Accompagner l'étudiant dans son apprentissage, accompagner le jeune chercheur vers son autonomie, accompagner son équipe de recherche ou son laboratoire dans son animation, son fonctionnement et sa stratégie, accompagner son école ou son université afin de répondre au mieux aux besoins et enjeux de recherche et de formation par la recherche. Enfin, je pense que le rôle du chercheur est également d'accompagner ses concitoyens par exemple via des actions de vulgarisation scientifique ou des tables rondes.

Deuxième partie

Contributions à l'axe : implémentation de fonctions cryptographiques

Dans cette partie, je présente les résultats obtenus dans le cadre de mes travaux de recherches dans le domaine de l'implémentation de fonctions cryptographiques. Dans une première section, je présente les travaux réalisés dans le cadre du chiffrement homomorphe. Dans une seconde section, je présente les travaux réalisés dans le cadre du chiffrement symétrique AES.

Pour chacune de ces sections, je donne une brève introduction au sujet d'étude puis, je détaille les principales contributions et enfin je fournis une sélection des principaux articles illustrant les travaux réalisés.

Les travaux décris dans cet axe correspondent à trois projets de thèse de doctorat que j'ai co-encadrés :

— Co-conception logicielle / matérielle pour l'accélération de schémas de chiffrement homomorphe (2014 - 2017)

— Réalisation d'un outil de prototypage et d'aide à l'exploration de paramètres des schémas de chiffrement homomorphe (2015 - 2018)

— Shuffling matériel pour l'algorithme de chiffrement AES dans un contexte IoT (2017-2021)

La figure 2 présente des mots clés associés à chacun de ces travaux de thèse. Cette représentation permet de mettre en évidence les liens entre ces trois projets.

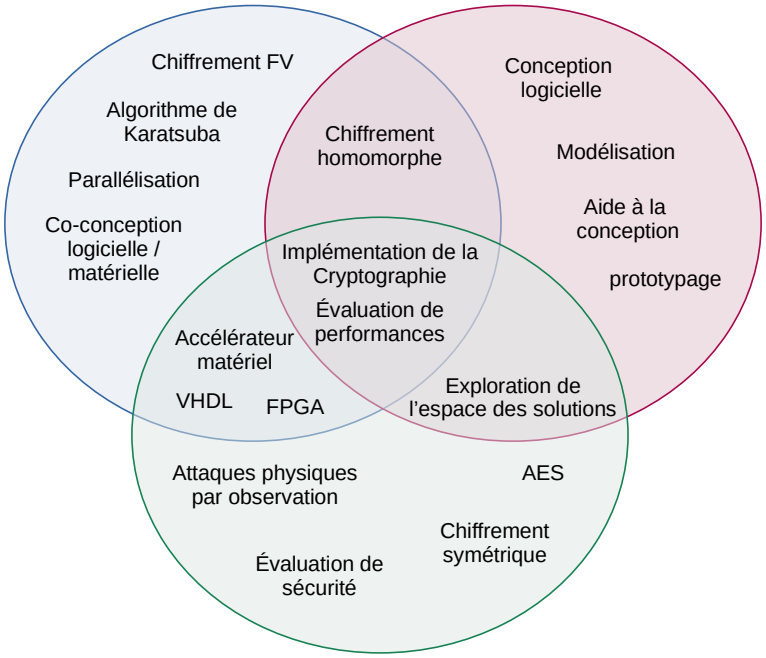


FIGURE 2 – Couverture thématique de l'axe implémentation de fonctions cryptographiques

Bien que l'implémentation de fonctions cryptographiques soit au cœur de ces différents travaux, l'évaluation des performances des solutions proposées est également un élément central. En effet, l'adoption des solutions proposées dépend en partie de leur performance en terme, par exemple, de débit, de latence, de fréquence, de surface, de coût mémoire, etc. Par conséquent, les travaux réalisés ont tous été évalués par ce prisme commun.

La figure 2 met également en lumière les liens directs entre deux projets. Deux projets ont été menés sur la thématique du chiffrement homomorphe. Le premier s'est intéressé à l'implémentation des primitives d'un schéma de chiffrement spécifique en s'appuyant sur une approche

de co-conception logicielle / matérielle. Le second s'est focalisé sur la conception d'un outil logiciel pour le prototypage et l'aide à l'exploration de paramètres des schémas de chiffrement homomorphe. Les deux projets ont permis d'adresser deux aspects complémentaires liés à l'implémentation de schémas de chiffrement homomorphe : proposer une solution performante pour la réalisation des traitements complexes et proposer un outils permettant de faire le lien entre les besoins applicatifs et les caractéristiques des schémas de chiffrement homomorphe (paramètres, empreinte mémoire, performances temporelles, etc.) en proposant une exploration automatisée de l'espace des solutions. De plus, bien que le premier projet se soit focalisé sur un schéma particulier (le chiffrement FV), il a permit d'enrichir le second via la fourniture d'information concernant le potentiel offert par l'accélération matérielle d'opérations communes à différents schémas pouvant être intégrés dans l'outil développé durant le second projet.

Le troisième projet s'est intéressé à la protection d'une implémentation AES bas coût face à des attaques physiques exploitant des canaux auxiliaires. L'approche proposée s'appuie sur la génération matérielle d'aléa pour *shuffler* les opérations réalisées. La figure 2 met en évidence deux liens avec les projets se focalisant sur le chiffrement homomorphe. Le premier concerne l'expertise en développement d'accélérateurs matériels sur cible FPGA. Le second concerne la volonté commune d'explorer l'espace des solutions via la variation de paramètres, de choix architecturaux et d'options de synthèse afin d'en analyser l'impact sur l'implémentation finale. Ce troisième projet a également ouvert, pour ma part, la thématique des attaques physiques communes avec certains travaux de l'axe de recherche "sécurité des processeurs embarqués".

5 Étude et implémentation de schémas de chiffrement homomorphe

5.1 Le chiffrement homomorphe

Le chiffrement homomorphe est un outil récent de la cryptographie moderne permettant la réalisation de traitements sur des données chiffrées. Ce type de chiffrement permet d'obtenir les mêmes résultats, après déchiffrement, que dans le cas où les traitements sont réalisés sur des données non chiffrées. Le chiffrement homomorphe permet donc le déport de calculs sur des données sensibles tout en garantissant la confidentialité de ces dernières malgré l'utilisation de machines appartenant à des tiers qui ne sont pas de confiances. La figure 3 illustre ce scénario.

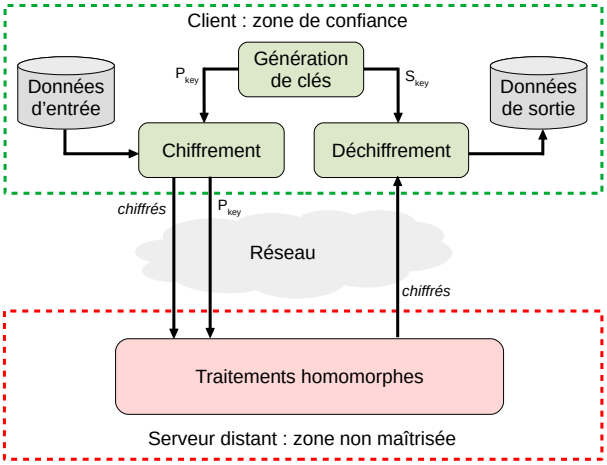


FIGURE 3 – Exemple de scénario d'utilisation du chiffrement homomorphe

Certains schémas cryptographiques usuels possèdent des propriétés homomorphiques pour l'addition [Paillier 1999] ou la multiplication [Elgamal 1985] (le lecteur pourra se référer à la référence [Fontaine 2007] pour une présentation de schémas dit partiellement homomorphes). En 2009, Greg Gentry, en s'appuyant sur la difficulté des problèmes sur les réseaux euclidiens, propose le premier schéma cryptographique complètement homomorphe (*Fully Homomorphic Encryption* FHE) [Gentry 2009a, Gentry 2009b] permettant la réalisation d'un nombre illimité d'opérations d'addition et de multiplication. Malheureusement, cette première approche s'avère être trop complexe pour une utilisation pratique. Cependant, ces travaux ont fourni à la communauté scientifique une base solide sur laquelle s'appuyer. En particulier, Gentry a introduit une structure intéressante et un mécanisme, appelé *bootstrapping* permettant la réduction du bruit accumulé lors de la réalisation des opérations d'addition et de multiplication sur les chiffrés. Suite à ces travaux, plusieurs schémas de chiffrement complètement homomorphe ont été proposés dans la littérature, mais aucun d'entre d'eux ne permettait une utilisation pratique. Certains de ces schémas sont construits sur la base de schémas dit *Somewhat Homomorphic Encryption* (SHE) permettant la réalisation d'un nombre arbitraire d'additions mais un nombre limité de multiplications. Pour cela, le mécanisme de bootstrapping est utilisé, alourdissant alors un système déjà complexe. Cependant, en sacrifiant la flexibilité offerte par les solutions FHE, les solutions SHE ont offert une approche plus réaliste pour la mise en oeuvre d'applications réelles.

Il est important de noter que le choix et l'utilisation pratique des schémas de chiffrement homomorphe est rendu difficile en raison de différentes contraintes. Tout d'abord, ces schémas, engendrent une expansion importante des chiffrés. En effet, le facteur d'expansion peut varier de quelques milliers à plusieurs centaines de milliers en fonction du schéma et du niveau de sécurité attendu. Cela est dû aux structures mathématiques utilisées et au fait que ces schémas doivent être probabilistes pour garantir des propriétés de sécurité. De plus, il est nécessaire de considérer

le pire cas en terme de complexité algorithmique des traitements réalisés sur les données chiffrées. Ces éléments participent au paramétrage de la structure mathématique sous-jacente d'un schéma de chiffrement homomorphe et, par conséquent, impacte le niveau de sécurité et le temps d'exécution. Enfin, le besoin de flexibilité de l'utilisateur influence les choix du développeur lors de l'implémentation d'une solution. Une première approche consiste à fixer un nombre maximum de multiplications réalisables sur les données chiffrées en prenant éventuellement en compte des évolutions applicatives futures. Le développeur peut alors se tourner vers une solution SHE. Une seconde approche consiste à s'appuyer sur une solution FHE supportant un nombre illimité de multiplications réalisables sur les données chiffrées.

Dans le but de compléter cette brève introduction aux schémas de chiffrement homomorphe, le lecteur trouvera dans l'article ci-dessous publié en 2017 [Bonnoron 2017], un focus sur les schémas SHE. Cet article présente certaines solutions de l'état de l'art et met en lumière les challenges accompagnant la mise en oeuvre de ces solutions. Dans les deux sections suivantes les travaux de recherche menés dans cette thématique sont présentés.

Somewhat/Fully Homomorphic Encryption: implementation progresses and challenges

Guillaume Bonnoron[1,2], Caroline Fontaine[2], Guy Gogniat[3], Vincent Herbert[2,4], Vianney Lapôtre[3], Vincent Migliore[3], and Adeline Roux-Langlois[5]

[1] Chair of Naval Cyber Defense, Ecole Navale - CC600, F-29240 Brest Cedex 9, France,
guillaume.bonnoron@ecole-navale.fr,
[2] CNRS and IMT Atlantique, UMR 6285, Lab-STICC, CS 83818, F-29238 Brest cedex 3, France,
caroline.fontaine@imt-atlantique.fr,
[3] Univ. Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France,
firstname.lastname@univ-ubs.fr,
[4] CEA LIST, Point Courrier 172, 91191 Gif-sur-Yvette Cedex, France,
vincent.herbert@cea.fr,
[5] CNRS - IRISA, Campus universitaire de Beaulieu 35042 Rennes, France,
adeline.roux-langlois@irisa.fr

Abstract. The proposed article aims, for readers, to learn about the existing efforts to secure and implement Somewhat/Fully Homomorphic Encryption ((S/F)HE) schemes and the problems to be tackled in order to progress toward their adoption. For that purpose, the article provides, at first, a brief introduction regarding (S/F)HE. Then, it focuses on some practical issues related to the adoption of (S/F)HE schemes, i.e. the security parameters, the existing implementations and their limitations, and the management of the huge complexity caused by homomorphic calculation. These issues are analyzed with the help of recent related work published in the literature, and with the experience gained by the authors through their experiments.

Keywords: Homomorphic Encryption, Data Privacy, Confidentiality, Security, Real World.

1 Introduction

Homomorphic Encryption (HE) is a recent promising tool in modern cryptography, that allows to carry out operations on encrypted data. The key idea is that performing some operations on encrypted data will provide the same result after decryption as if the computation would have been performed on the original plain data. Then, with such a tool one could outsource storage and/or computation without endangering data's privacy. Figure 1 illustrates different client/server scenarii benefiting from homomorphic encryption. Some examples of applications can be found in the literature, *e.g.* [NLV11,GLN12,LLN14,BPB09,CGGI16b]. While usual cryptographic schemes sometimes have homomorphic properties, for addition [Pai99] or multiplication [ElG85] operations (see also [FG07] for a survey on such partially homomorphic schemes), an important breakthrough has been made in 2009 according to the work of Gentry [Gen09b,Gen09a]. Based on hard lattice problems, he proposed the first *Fully Homomorphic Encryption* (FHE) scheme, enabling to perform an unlimited number of additions and multiplications secretly. Unfortunately, this scheme was too complex to be used in practice. Nevertheless, it introduced an interesting structure as well as a nice trick called *bootstrapping* to reduce the inherent noise that accompany the running of additions and multiplications. Following this seminal work, several FHE have been proposed, but none of them were usable in practice. It is interesting to notice that some FHE are related to more practical schemes called *Somewhat Homomorphic Encryption (SHE) schemes*, that enable an arbitrary number of additions but a bounded number of multiplications. In fact, with the help of bootstrapping, one can design FHE schemes from SHE schemes. Nevertheless, this bootstrapping step adds an extra cost to an already quite heavy process.

To address a particular use case, one must have in mind several constraints that will be crucial for choosing the right homomorphic encryption solution. First, using (S/F)HE schemes will lead to a huge ciphertext expansion (say from 2.000 to 500.000 or even 1.000.000 according to the

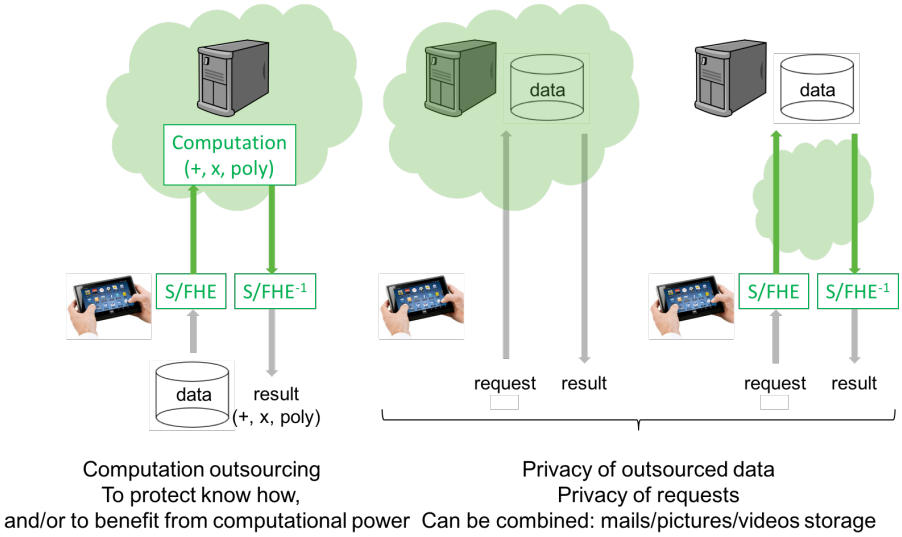


Fig. 1. A need for processing encrypted data, to ensure privacy in outsourced computation, outsourced storage and databases requests. Green areas show what is encrypted.

scheme and the targeted security level). This is due to the fact that homomorphic schemes must be probabilistic to ensure semantic security, and to the particular underlying mathematical structures. Second, as we will see later in this paper, we will need to consider only worse case complexity for the algorithms that will bu run on the encrypted data; also considering that the underlying operations are intrinsically expansive, this will drastically penalize the global running time. These are the most important constraints we have to address. The underlying common point behind these remarks concerns the targeted security level, as it determines the parameters used to use the mathematical appropriate structure that will enable a coherent computation, and then the ciphertext expansion and the running time. Another important parameter concerns the strategy of the developer in terms of flexibility.

Hence, two strategies can be followed to drive the choice. On one hand one can fix a maximum multiplicative depth for the Boolean circuit to be evaluated on the encrypted data. This may take into account a small or at least bounded flexibility for future modifications of the circuit to be evaluated. In this case, SHE is the best choice. On the other hand one may want to be able to use any Boolean circuit and then to handle an unbounded multiplicative depth. In this case, FHE is the only choice.

In this paper, we will focus on SHE schemes because they are the most promising today, and we will discuss their implementations issues. We will also provide some information on the state-of-the-art concerning the use of bootstrapping to modify these schemes intro FHE ones. Our goal is to provide the reader the best starting points to handle the complexity of the issues to address and the efforts made to make these schemes become sufficiently efficient to be used in practice.

2 Existing (F/S)HE solutions

Due to lack of space, we do not provide any precise description and let the reader refer to the mentioned papers to get all the mathematical details related to the schemes.

2.1 SHE from classical crypto world

The first scheme that enabled to perform both additions and multiplications is due to Boneh *et al.* [BGN05]. This pairing-based SHE enables to perform as many additions as wanted, but only one multiplication. Hence, it is not really flexible.

We have worked recently [HF17] on the design of a variant of this scheme that allows to handle multiplicative depth 2 (instead of 1). Our solution employs together two improvements of the original scheme, based on [Fre10,CF15]. We will refer to it as BGN2 in the rest of the document.

2.2 Lattice-based (S/F)HE

Whereas the first FHE scheme has been proposed by Gentry in [Gen09a], the first SHE based on lattices has been propose by Aguilar *et al.* in [AMGH10]. These schemes have been followed by many others. First generation encompass the older ones, like [vDGHV10,SV10,GHS12a,GH11]. Second generation [BGV12,CNT12,BGV12,Bra12,FV12] started with leveled SHE schemes based on modulus switching, which have then been improved to remain scale invariant, etc. They were followed by third generation schemes like [GSW13,BLLN13,BV14,KGV15,DS16]. In fact several of these schemes consist of improvement of previous ones, and the genealogy in rich of cross-fertilization. For example, SHIELD [GSW13] and F-NTRU [DS16] are the third generation schemes equivalent to, respectively, FV [FV12] and YASHE' [BLLN13]. With larger costs for the first homomorphic multiplications, these schemes have a much better asymptotic behavior.

3 Customization and optimization of both program and data

The issues discussed in this section have been addressed in few papers only [AMFF$^+$13,FSF$^+$13,CDS15] for the moment, but they are really important to handle real deployment of this technology.

3.1 Program management

To establish a proper link between the program we want to run and (S/F)HE schemes, one must rely on the fact that any program can be expressed as a Boolean circuit involving `XOR` and `AND` gates, and that `XOR` and `AND` are precisely the addition and multiplication operators over bits. This being said, the game is to express the program through such a Boolean circuit, and to optimize it. The last step will then be to evaluate this optimized Boolean circuit over the encrypted data while replacing `XOR` and `AND` gates by the corresponding encrypted operators provided by a (S/F)HE library.

The optimization step of the Boolean circuit is crucial, as it will give us the multiplicative depth we will have to handle with the (S/F)HE scheme. This will have an impact on the parameters of this scheme (*e.g.* size of the lattice, modulus, etc), and then on the ciphertexts size and on the resulting computation time. Usally, we focus on this multiplicative depth, but we also have to be careful that a very large amount of additions may also lead to heavy computations. Also, sometimes additions following multiplications may have a different impact than additions occurring at lower multiplicative depth. Also, when using leveled (S/F)HE using modulus switching techniques like [BGV12], one has to optimize at which level each ciphertext stands, to avoid any extra and costly modulus switching. At last, the noise growth is usually symmetric between the left and right operands, but for some particular recent schemes like [GSW13] the noise growth is asymmetric over the multiplications. In this case, it is important to optimize which operand will be left or right. All these aspects should be taken into account. This optimization issue has not been sufficiently addressed in the literature, and a lot of work remains to be done to set theoretical bounds and practical strategies to handle such an optimality.

Another issue related with the customization of the Boolean circuit is that it may contain `if-then-else` or `repeat...until` expressions leading to branches with dynamic size depending on the processed data. To handle such statements properly, on must have in mind that the processed data they are depending on are encrypted. Moreover, it is crucial that during the running process no information about the real value of the underlying data may leak. To handle this is easy but costly. In fact, the best approach is to rewrite such parts of the program with the help of Boolean expressions. If we look at the `if-then-else` example, one can rewrite `if c then x=a else x=b` as `x = (c AND a) XOR ((NOT c) AND b)`. The whole expression will then be evaluated over the encrypted data, and the final encrypted result will be the good one without revealing anything on the plain value of `c`. Unfortunately the price to be paid is high, as we always have to evaluated the whole expression, meaning that we always need to evaluate the deepest branch of the tree. Hence, computing over encrypted data always requires the worse-case complexity. This means that when choosing an algorithm to perform a particular computation over encrypted data, one must choose the algorithm that provides the best worse-case complexity (whereas usually we look for the best average-case complexity).

Finally, as (S/F)HE remain costly today, on must be really careful not to perform non necessary heavy computations. According to a particular applicative scenario, one must try to perform as much classical encryption as possible, and to think of some pre-processing over the plaintexts that may help to enlight the computation over the ciphertexts.

3.2 Data management

Some schemes work only over integers, and some over bits. And some can manage both. If the application scenario does not require data depend behavior, then working over integers is usually the best choice. But if the application scenario requires some `if-then-else` or `repeat...until` like statements, then we need to work at the bit level to be able to perform `<`, `>` and `=` operators. In this case, we have to provide home maid basic operators for integers additions and multiplications, floats management,... but this is the price to pay to handle such dynamic behaviors properly.

Some schemes may also be compliant with batching ability, then enabling to group several pieces of data on the same plaintext, typically a polynomial, and to process all these pieces of data at the same time. This drastically reduces the costs in terms of space (memory) and running time, as several plaintexts (resp. ciphertexts) are processed in one shot . In the literature, batching is usually addressed through SIMD and RNS, see for example [SV14,BEHZ16].

4 Flexibility of (S/F)HE schemes in terms of multiplicative depth management

4.1 SHE from classical crypto world

SHE schemes coming from classical crypto world are not flexible. Boneh-Goh-Nissim [BGN05] scheme only enables the evaluation only of degree 2 polynomials, whereas our extension BGN2 enables the evaluation of degree 4 polynomials [HF17].

4.2 Lattice-based (S/F)HE schemes

SHE schemes based on these lattice problems are generally much more flexible and can be turned into fully homomorphic schemes allowing computation with arbitrary depth. The downside for this flexibility is the increased size of ciphertexts and keys, leading to heavier computations. For these schemes, one have to choose the maximum multiplicative depth we want to be able to handle before deployment, as this multiplicative depth will drive the parameters determining the underlying lattice. Once the scheme is deployed, it cannot be modified. Moreover, as we have to handle "the higher the multiplicative depth, the higher the memory and time costs" the parameters much be chosen very carefully to maintain an acceptable cost while ensuring sufficient flexibility for future process.

FHE schemes are generally heavier to deploy, as they need the bootstrapping step to be sure to enhance the noise growth properly whatever the multiplicative depth. Hence, they should be used only if one does not know in advance the multiplicative depth we would like to handle in the future in the targeted application scenario. Good recent works to better understand the limits of this solution are [PV15], which provides a way to optimize the bootstraps management (for any FHE), and [CGGI16a] that proposes the more efficient current way to execute bootstrapping (especially for FHE based on [GSW13]).

5 Security of (S/F)HE schemes

By nature, these schemes have been designed to enhance privacy and security. Hence, the analysis of their security is crucial. The best security level that has been achieved for such schemes is IND-CPA. We know that IND-CCA2 in not achievable, and the design of efficient schemes achieving IND-CCA1 is still open. Now, we will split the security study according to the underlying mathematical rationales.

5.1 SHE from classical crypto world

Schemes which are based on mathematical objects which have already been deeply studied by the cryptographic community are better understood, and their security is easier to manage. Among such schemes, one can mention partially homomorphic schemes as ElGamal, Paillier, and their variants, which are out of the scope of this article as they cannot handle at the same time additions and multiplications. As mentioned in Section 2.2, one can also mention Boneh-Goh-Nissim scheme [BGN05], a pairing-based SHE scheme which enables to perform as many additions as wanted, but only one multiplication, and our scheme called BGN2 which can handle one more multiplication depth [HF17].

The security of BGN2 is based on the generalized subgroup decision assumption[6]. This problem is derived from the decision Diffie-Hellman assumption [Bon98]. Two possible choices to instantiate groups are to select either an elliptic curve or an hyperelliptic curve. We place ourselves in the first case, where the security assumption reduces to the elliptic curve discrete logarithm problem. The recommended group size is given by different academic and private organizations at www.keylength.com according standard security levels. The order of ciphertext expansion in BGN2 is thousands.

5.2 Lattice-based (S/F)HE

The *Learning With Errors* problem is about solving a system of several *noisy* linear equations. Its ring variant, *Ring-LWE*, allows to design more efficient schemes with faster computations and smaller keys and ciphertexts. These problems attract large attention, beyond HE, because they are believed to be quantum resistant: no known quantum algorithms perform better than the classical ones against them.

These schemes were introduced in the previous decade, i.e. quite recently in the time scale of cryptography, and one must tell that there is still a gap between the theoretical hardness proofs and the practical behavior of the known attacks.

Formal proofs The first FHE scheme of Gentry [Gen09b] was based on two assumptions, the *Bounded Distance Decoding problem* and the *Sparse Subset Sum problem* which are not standard in lattice-based cryptography. The first scheme proven secure under the LWE assumption is proposed by Brakerski and Vaikuntanathan in 2011 [BV11]. This result was followed by numerous constructions based on LWE and Ring-LWE (as [Bra12,BGV12]...).

The LWE and Ring-LWE problem are proven to be at least as hard as well-known hard problems (in their worst-case) on lattices (respectively on ideal-lattices). Another advantage of recent lattice-based schemes is that their security is proven under those assumptions.

Experimental security evaluation Even at this point, concrete security behind homomorphic encryption is still moving. Thus, extracting realistic parameters for a given security level is a main challenge of (S/F)HE scenario. The standard approach so far is to focus on concrete attack means. Building on surveys about existing attacks against LWE [APS15] and Ring-LWE [Pei16], experiments must be pushed further to provide experimental results of secured parameters for most promising homomorphic schemes.

Among these most promising schemes one can mention YASHE' [BLLN13], FV [FV12] and SHIELD [GSW13]. YASHE' and FV have been published almost at the same time, but YASHE' took benefit from a strong lobby and became more popular in the proposed implementations. However, confidence in this scheme has been recently damaged by the subfield/sublattice attack [ABD16,KF16]. Making YASHE' immune to these attacks would lead to oversize its parameters, far too much for practical use [DGBL+15]. This is why among second generation schemes FV is now the real challenger, and has received a lot of attention during the past months [LCP16,Cry16,BEHZ16].

[6] We choose to employ asymetric pairings to compute homomomorphic product of fresh ciphertexts. The use of symmetric pairings would change the computational hardness assumption[Fre10, page 46].

6 Existing implementations of (S/F)HE schemes

6.1 Software implementations

We implemented BGN2, which have been presented and briefly discussed above. In this cryptosystem, the homomorphic multiplication asks to compute pairings. We chose to compute an optimal Ate pairing over an elliptic curve in the Barreto-Naehrig curve family [BN05] using a program called DCLXVI [NNS10]. This work is not yet published, but will be available soon both as an article and as a software.

Concerning lattice-based (S/F)HE schemes, several implementations have emerged since their introduction in 2009. Due to the pace of evolution in the theoretical field, some are now outdated but served as good proof of concept in the early days of (S/F)HE ([Bre,Cor,Lep]). Other libraries ([DGBL$^+$15,Hal,Cry16,LCP16]) implement the latest techniques described in [SV10,FV12,BGV12,BLLN13]. There are also private implementations like those used in [AMFF$^+$13,FSF$^+$13,CDS15,BEHZ16]. Most of the libraries aim at providing tools for experiments to the academic community, except [DGBL$^+$15,LCP16,CDS15] which can be used as building blocks for more industrialized developments.

Today, no complete benchmark is available to provide a fair and complete overviews of the efficiency of all these schemes. The reader can refer to [LN14] for a comparison of FV and YASHE', and to [MBF16] for a first discussion and comparison of FV, SHIELD and F-NTRU in terms of pros and cons, and parameters setting.

6.2 Hardware implementations

Hardware implementation is one of the two principal ways to accelerate FHE/SHE schemes with dedicated components. The second one is the GPU acceleration. Even if performances using GPU are very scheme-dependent, it can be a good alternative to set up an homomorphic server quickly with acceptable performances [KGV15].
Hardware accelerators focus on accelerating the most complex operation of Homomorphic Encryption Schemes, the multiplication of homomorphic operands. Depending on the scheme, a million-bit integer multiplier or a polynomial of degree $n \in [4096, 32768]$ with coefficients of size $\log_2 q \in [125, 1228]$ is required. In [DOS13], an ASIC implementation of million-bits multiplier performs the multiplication operation in 7.74 ms using NTT algorithm. This computation time corresponds to the computation time on a Xeon, but can be implemented as a co-processor and thus requiring a much smaller area. For polynomial based homomorphic encryption, due to the fact that one must address various size of polynomials, different architectures have been investigated. To our knowledge, all implementations are based on NTT algorithm too, except in [MMRL$^+$17] for small size polynomials which implements Karatsuba algorithm instead. In [PNPM], a usual but optimized NTT implementation is presented for two parameter sets. The proposed accelerator performs an homomorphic multiplication in 6.5 ms for $n = 4096$ and $\log_2 q = 125$ bits, and 48 ms for parameters $n = 16384$ and $\log_2 q = 512$ bits. Authors of [PNPM] implemented 512×512 bits multipliers with a small modular reduction by selecting a Solinas prime modulus [Sol11]. Due to the size of polynomials and coefficients, a cache is implemented to connect the external memory used to store intermediate coefficients. They also reported a bottleneck due to the divide and rounding required by YASHE' scheme, especially for large integers. That is why in [SRJV$^+$] a precomputation is performed on polynomials to reduce the size of coefficients. They split a ciphertext into a few polynomials by using the *Chinese Reminder Theorem* (CRT) on each coefficient. The overall architecture is based on an array of crypto-units, which gives some flexibility to process several residue polynomials in parallel. For parameters $n = 32768$ and $\log_2 q = 1228$ bits, their accelerator performs an homomorphic multiplication in 121 ms including 25 ms spent for CRT.
Table 1 summarize the different hardware implementations available for both integer based and polynomial based Homomorphic Encryption Schemes.

7 How to handle such a huge complexity and expansion?

(S/F)HE schemes defined on Elliptic Curves and Pairing present a smaller complexity and expansion, and their security level is quite clear, but are very limited in terms of multiplicative depth.

Table 1. Timing results for the hardware implementation of Homomorphic Encryption.

Integer based Homomorphic encryption					
Scheme	Algorithm	Size	Homomorphic Encryption	Homomorphic Multiplication	Work
Gentry-Halevy	NTT	1 M bits	2.09 s	7.74 ms	[DOS13]
DHGV		19 M bits	3.9 s	*no results*	[CMO+]

Polynomial based Homomorphic encryption						
Scheme	Algorithm	n	$\log_2 q$	Homomorphic Encryption	Homomorphic Multiplication	Work
YASHE'	Karatsuba	2560	124 bits	*not implemented*	4.73 ms	[MMRL+17]
	NTT	4096	125 bits		6.5 ms	[PNPM]
		16384	512 bits		48 ms	
		32768	1228 bits		121 ms	[SRJV+]

For some applications this may be sufficient, and particularly interesting, this is why we discussed them here.

Nevertheless, the biggest hope for the future comes from lattice-based schemes, which promise to handle larger multiplicative depths processing. Once their security will be better understood, their main drawbacks are their algorithmic complexity and the related ciphertext expansion. This is why it is important to pursue designing new schemes and to look for lighter solutions. Current ciphertext expansions can go from $10,000$ up to $1,000,000$, depending on the schemes and on the parameters that have been chosen (and which are directly related with the targeted security level). The encrypted data must be uploaded from the client device to the server, then processed on the server, and finally the encrypted result must be downloaded from the server to the client device. Hence, its size is critical.

To reduce the size of the uploaded data on the first step, [NLV11] proposed to combine the (S/F)HE scheme with symmetric encryption schemes. The main idea is that the data to be uploaded will be encrypted by the symmetric encryption scheme, and then sent to the server without any size expansion. Hence, the server will trans-crypt this encrypted data to produce a new ciphertext which corresponds to the encryption of the same data with the (S/F)HE that will be used on the server to perform the desired computation. This trans-cryption step will require the decryption circuit of the underlying symmetric encryption scheme to be evaluated by the (S/F)HE scheme. This step's complexity is critical, and will lead the choice of the symmetric cipher to use. Following this idea, several symmetric encryption schemes have been investigated. We will first mention on-the-shelf block ciphers like AES [GHS12b,CCK+13,DHS14], and the lightweight block ciphers Simon [LN14] and Prince [DSES14]. But the evaluation of these ciphers by the (S/F)HE remains too complex, and recent papers proposed new ciphers designed specifically for this purpose (*i.e.* with a low multiplicative depth): the block cipher Low-MC [ARS+15], which has been broken [DLMW15] and patched [Rec16]; the stream cipher Kreyvium [CCF+16], whose security is the same as the well-studied stream cipher Trivium; and a more recent stream cipher proposal called FLIP [MJSC16], which should be used carefully [DLR16]. These papers include experimental material and results. More exotic solutions are discussed in [FHK16], but without experimental data in the paper.

The second way to reduce ciphertexts weight is to pack several inputs on the same plaintext structure through batching. This has been briefly discussed in Section 3.2

8 Acknowledgement

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 643964.

9 Conclusion

There is still a lot of work to be done, but everything is moving fast and recent progress are quite impressive. Hence, for some applications which are not too critical in terms of memory and time

costs it is time to adopt a practical approach to make dream become reality. How to choose the good (S/F)HE scheme for a given application scenario, and how to set it up in the best way? The answer is not trivial at all, and this article provides some hints concerning the implementation issues of these promising but still tricky and heavy schemes. Our goal was to share our discussions and reflections about all the identified issues, and to provide ad-hoc references to help the reader in his exploration of a very prolific and dense literature. It is clear that more comparisons should be performed between the most promising schemes. A few papers compared two schemes at a time, like [LN14], and a first attempt to provide a wider analysis can be found in [MBF16]. But it is clear that it should be pushed further, and that fair benchmarks based on available implementations have to be driven. Moreover, a fair and precise comparison should also be driven to properly compare SHE schemes coming from classical crypto world with lattice-based ones when targeting a small multiplicative depth, in terms of time and space complexity. At the same time, open issues remain concerning a precise evaluation of the practical security of lattice-based schemes, as well as on the optimization of the Boolean circuits we want to evaluate over the encrypted data.

References

ABD16. M. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched ntru assumptions: Cryptanalysis of some fhe and graded encoding schemes. Cryptology ePrint Archive, Report 2016/127, 2016.

AMFF+13. Carlos Aguilar-Melchor, Simon Fau, Caroline Fontaine, Guy Gogniat, and Renaud Sirdey. Recent advances in homomorphic encryption: A possible future for signal processing in the encrypted domain. *IEEE Signal Processing Magazine*, 30(2):108–117, 2013.

AMGH10. C. Aguilar Melchor, P. Gaborit, and J. Herranz. Additively homomorphic encryption with d-operand multiplications. In *Annual Cryptology Conference*, pages 138–154. Springer, 2010.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

ARS+15. Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

BEHZ16. Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes. Cryptology ePrint Archive, Report 2016/510, 2016. `http://eprint.iacr.org/2016/510`.

BGN05. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

BGV12. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proc. of the 3rd Innovations in Theoretical Computer Science Conference – ITCS 2012*, pages 309–325. ACM, 2012.

BLLN13. J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Proc. of Cryptography and Coding: 14th IMA International Conference – IMACC 2013*, pages 45–64. Springer, 2013.

BN05. Paulo S. L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2005.

Bon98. Dan Boneh. The Decision Diffie-Hellman Problem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, pages 48–63, London, UK, UK, 1998. Springer-Verlag.

BPB09. Tiziano Bianchi, Alessandro Piva, and Mauro Barni. On the implementation of the discrete Fourier transform in the encrypted domain. *IEEE Transactions on Information Forensics and Security*, 4(1):86–97, 2009.

Bra12. Z. Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Proc. of CRYPTO*, volume 7417 of *LNCS*, pages 868–886. Springer, 2012.

Bre. Michael Brenner. Hcrypt project. Available at `http://www.hcrypt.com`.

BV11. Z. Brakerski and V. Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *Proc. of FOCS*, pages 97–106, 2011.

BV14. Z. Brakerski and V. Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proc. of the 5th Conference on Innovations in Theoretical Computer Science – ITCS 2014*, pages 1–12. ACM, 2014.

CCF+16. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrede Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. In *Fast Software Encryption 2016*, Fast Software Encryption 2016, Bochum, Germany, March 2016. Springer Verlag.

CCK+13. Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch Fully Homomorphic Encryption over the Integers. In *EUROCRYPT*, volume 7881 of *LNCS*, pages 315–335. Springer, 2013.

CDS15. Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 13–19. ACM, 2015.

CF15. Dario Catalano and Dario Fiore. Using Linearly-Homomorphic Encryption to Evaluate Degree-2 Functions on Encrypted Data. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1518–1529. ACM, 2015.

CGGI16a. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds*, pages 3–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

CGGI16b. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic LWE based e-voting scheme. In *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, pages 245–265, 2016.

CMO+. Xiaolin Cao, Ciara Moore, Maire O'Neill, Neil Hanley, and Elizabeth O'Sullivan. High-Speed Fully Homomorphic Encryption over the Integers. In *Proc. of the Workshop on Encrypted Computing and Applied Homomorphic Cryptography – WAHC 2014*.

CNT12. Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2012*, pages 446–464. Springer, 2012.

Cor. Coron, Jean-Sébastien. An implementation of the DGHV fully homomorphic scheme. Available at `https://github.com/coron/fhe`.

Cry16. CryptoExperts. FV-NFLlib. Available at `https://github.com/CryptoExperts/FV-NFLlib`, 2016.

DGBL+15. Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Manual for Using Homomorphic Encryption for Bioinformatics. Technical Report MSR-TR-2015-87, November 2015.

DHS14. Yarkin Doröz, Yin Hu, and Berk Sunar. Homomorphic AES Evaluation using NTRU. *IACR Cryptology ePrint Archive*, 2014:39, 2014.

DLMW15. Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized Interpolation Attacks on LowMC. *IACR Cryptology ePrint Archive*, 2015:418, 2015.

DLR16. Sébastien Duval, Virginie Lallemand, and Yann Rotella. Cryptanalysis of the FLIP Family of Stream Ciphers. *IACR Cryptology ePrint Archive*, (271), 2016.

DOS13. Yakin Doroz, E Ozturk, and Berk Sunar. Evaluating the Hardware Performance of a Million-Bit Multiplier. In *Proc. of Euromicro Conference on Digital System Design – DSD 2013*, 2013.

DS16. Y. Doröz and B. Sunar. Flattening ntru for evaluation key free homomorphic encryption. Cryptology ePrint Archive, Report 2016/315, 2016.

DSES14. Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth, and Berk Sunar. Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince. In *WAHC*, volume 8438 of *LNCS*, pages 208–220. Springer, 2014.

ElG85. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

FG07. C. Fontaine and F. Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP J. Inf. Secur.*, 2007(1):1–15, 2007.

FHK16. Pierre-Alain Fouque, Benjamin Hadjibeyli, and Paul Kirchner. *Homomorphic Evaluation of Lattice-Based Symmetric Encryption Schemes*, pages 269–280. Springer International Publishing, Cham, 2016.

Fre10. David Mandell Freeman. Converting Pairing-Based Cryptosystems from Composite-Order Groups to Prime-Order Groups. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 44–61. Springer, 2010.

FSF+13. Simon Fau, Renaud Sirdey, Caroline Fontaine, Carlos Aguilar-Melchor, and Guy Gogniat. Towards practical program execution over fully homomorphic encryption schemes. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 284–290. IEEE, 2013.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

Gen09a. C. Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009.

Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

GH11. Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 107–109. IEEE, 2011.

GHS12a. Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology–EUROCRYPT 2012*, pages 465–482. Springer, 2012.

GHS12b. Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.

GLN12. Thore Graepel, Kristin E. Lauter, and Michael Naehrig. ML Confidential: Machine Learning on Encrypted Data. In *ICISC*, volume 7839 of *LNCS*, pages 1–21. Springer, 2012.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

Hal. Shai Halevi. HElib. Available at `https://github.com/shaih/HElib`.

HF17. Vincent Herbert and Caroline Fontaine. Software implementation of 2-depth pairing-based homomorphic encryption scheme. Cryptology ePrint Archive, Report 2017/091, 2017. `http://eprint.iacr.org/2017/091`.

KF16. Paul Kirchner and Pierre-Alain Fouque. Comparison between Subfield and Straightforward Attacks on NTRU. Cryptology ePrint Archive, 2016/717, 2016.

KGV15. Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Transactions on Computers*, PP(99):1–1, 2015.

LCP16. Kim Laine, Hao Chen, and Rachel Player. Simple encrypted arithmetic library - seal (v2.1). Technical report, September 2016.

Lep. Lepoint, Tancrède. A proof-of-concept implementation of the homomorphic evaluation of SIMON using FV and YASHE. Available at `https://github.com/tlepoint/homomorphic-simon`.

LLN14. Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private Computation on Encrypted Genomic Data. In *LATINCRYPT*, LNCS, 2014.

LN14. Tancrède Lepoint and Michael Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *AFRICACRYPT*, volume 8469 of *LNCS*, pages 318–335. Springer, 2014.

MBF16. Vincent Migliore, Guillaume Bonnoron, and Caroline Fontaine. Determination and Exploration of Practical Parameters for the Latest Somewhat Homomorphic Encryption (SHE) Schemes. working paper or preprint, October 2016.

MJSC16. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. *Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts*, pages 311–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

MMRL+17. Vincent Migliore, Maria Mendez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, and Guy Gogniat. Hardware/Software co-Design of an Accelerator for FV Homomorphic Encryption Scheme using Karatsuba Algorithm. *Accepted to IEEE Transactions on Computers*, 2017.

NLV11. Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *ACM CCSW*, pages 113–124. ACM, 2011.

NNS10. Michael Naehrig, Ruben Niederhagen, and Peter Schwabe. New Software Speed Records for Cryptographic Pairings. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *Progress in Cryptology - LATINCRYPT 2010, First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, Proceedings*, volume 6212 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2010.

Pai99. Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of Advances in Cryptology — EUROCRYPT 1999*, number 1592 in LNCS, pages 223–238, 1999.

Pei16. Chris Peikert. How (not) to instantiate Ring-LWE. In *International Conference on Security and Cryptography for Networks*. Springer, 2016.

PNPM. Thomas Pöppelmann, Michael Naehrig, Andrew Putnam, and Adrian Macias. Accelerating Homomorphic Evaluation on Reconfigurable Hardware. In *Proc. of Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 143–163. Springer.

PV15. Marie Paindavoine and Bastien Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 25–43, 2015.

Rec16. Christian Rechberger. The FHEMPCZK-Cipher Zoo. Presented at the FSE 2016 rump session, 2016.

Sol11. Jerome A. Solinas. Generalized Mersenne Prime. In *Encyclopedia of Cryptography and Security*, pages 509–510. Springer, 2011.

SRJV$^+$. Sujoy Sinha Roy, Kimmo Järvinen, Frederik Vercauteren, Vassil Dimitrov, and Ingrid Verbauwhede. Modular Hardware Architecture for Somewhat Homomorphic Function Evaluation. In *Proc. of Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 164–184. Springer.

SV10. Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pages 420–443. Springer, 2010.

SV14. N. P. Smart and F. Vercauteren. Fully homomorphic simd operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.

vDGHV10. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

5.2 Co-conception logicielle / matérielle pour l'accélération de schémas de chiffrement homomorphe

Entre 2014 et 2017, j'ai participé à l'encadrement des travaux de thèse de Vincent Migliore [Migliore 2017]. Ces travaux ont permis d'étudier et de développer une approche basée sur la co-conception logicielle / matérielle dans le but d'accélérer les traitements de schémas de chiffrement homomorphe. Dans la suite je présente les principales contributions issues de ces travaux.

5.2.1 Le schéma de chiffrement homomoprhe FV

Ces travaux se sont focalisés sur le schéma de chiffrement homomorphe FV [Fan 2012]. Ce schéma, basé sur une application directe du schéma [Brakerski 2012] au problème R-LWE, a été choisi car il faisait partie des schémas les plus populaires durant la période du projet de thèse.

FV repose sur le problème R-LWE (*Ring-Learning With error*), version polynomiale du problème LWE (*Learning With error)* dont l'application à l'homomorphe a été proposée dans [Regev 2009]. L'intérêt de ces schémas est que la clé publique ainsi que les chiffrés sont du même ordre de grandeur. En fonction du niveau de sécurité souhaité et de l'application cible, la clé publique ainsi qu'un chiffré peuvent aller de quelques centaines de milliers de bits, à plusieurs dizaines de millions de bits. De plus, FV possède les caractéristiques suivantes :

— Un chiffré est composé de seulement deux polynômes, l'addition homomorphe étant une simple addition terme-à-terme des polynômes des chiffrés, et la multiplication homomorphe est proche d'une multiplication naturelle des chiffrés (hors opération de relinéarisation réalisée quasiment systématiquement après chaque multiplication homomorphe afin de redonner au chiffré sa forme initiale)

— La génération des clés ne demande pas de polynômes avec des propriétés particulières. Cela évite tout problème de rejet de polynômes, coûteux en temps de calcul

— FV est capable d'adapter ses paramètres (degré et taille de ses coefficients) à toute profondeur multiplicative, permettant ainsi de minimiser sa complexité pour une application donnée.

5.2.2 Algorithme de Karatsuba

La multiplication polynomiale est l'opération élémentaire particulièrement complexe dans le schéma de chiffrement homomorphe FV. Durant ces travaux, nous nous sommes intéressés à l'accélération des primitives de FV pour des profondeurs multiplicatives ≤ 8. En particulier, nous avons étudié l'utilisation de l'algorithme de Karatsuba [Karatsuba 1963] pour réaliser les opérations de multiplication car il possède de nombreux avantages par rapport à la NTT [Pollard 1971] classiquement utilisée dans l'état de l'art :

— Son implémentation est simple, avec des pré- et post-traitements uniquement composés d'additions et de soustractions de polynômes

— Il engendre moins de restrictions sur le choix du module q et du degré n du polynôme cyclotomique nécessaire à la définition d'une instance du problème R-LWE

— Il est naturellement compatible avec le *batching*[5] car Karatsuba ne demande pas de changement d'espace comme le demande la NTT

— Il se prête bien à une application de type co-conception, avec un partage des calculs entre le logiciel et le composant matériel

L'algorithme de Karatsuba (en $O(n^{1.58})$) est une amélioration de l'algorithme de multiplication classique (en $O(n^2)$) visant à réduire le nombre de multiplications élémentaires nécessaires.

5. technique permettant de grouper plusieurs données dans le même texte clair / chiffré dans le but de paralléliser les traitements

Dans la suite, l'algorithme de Karatsuba est présenté pour la multiplication entière et polynomiale. Pour la version entière, les entiers sont découpés par morceaux de β bits. Pour une multiplication $c = a \cdot b$, on définit donc :

$$a = \sum_{i=0,}^{l-1} a_i \cdot 2^{\beta i}$$
$$b = \sum_{i=0,}^{l-1} b_i \cdot 2^{\beta i}$$

avec,

$$l = \lceil log_2 a/\beta \rceil$$
$$a_i = a_{(i\cdot\beta \to (i+1)\cdot\beta - 1)}$$
$$b_i = b_{(i\cdot\beta \to (i+1)\cdot\beta - 1)}$$

a_i et b_i représentent les chiffres de a et b dans la base β. Pour la version polynomiale, on suppose que les polynômes utilisés sont de degré n, donc pour une multiplication :

$$C = A \cdot B = \sum_{i=0,}^{2n} c_i X^i$$
$$A = \sum_{i=0,}^{n} a_i X^i$$
$$B = \sum_{i=0,}^{n} b_i X^i$$

Pour l'algorithme de Karatsuba, il est nécessaire de découper les opérandes d'entrée en deux parties. Pour simplifier, on supposera qu'ils sont de même longueur (cela revient à fixer $l = 2$, donc $\beta = \lceil log_2 a/2 \rceil$). On obtient alors $a = a_0 + a_1 \cdot 2^\beta$ et $b = b_0 + b_1 \cdot 2^\beta$. En utilisant l'algorithme de multiplication classique, on obtient :

$$c = a_0 b_0 + (a_0 b_1 + a_1 b_0) \cdot 2^\beta + a_1 b_1 \cdot 2^{2\beta}$$

En posant :

$$z_0 = a_0 b_0$$
$$z_1 = a_0 b_1 + a_1 b_0$$
$$z_2 = a_1 b_1$$

alors, $c = z_0 + z_1 2^\beta + z_2 \cdot 2^{2\beta}$.

L'optimisation proposée par Karatsuba consiste à remarquer que le terme z_1 peut s'écrire :

$$\begin{aligned} z_1 &= a_0 b_1 + a_1 b_0 \\ &= (a_0 + a_1) \cdot (b_0 + b_1) - a_0 b_0 - a_1 b_1 \\ &= (a_0 + a_1) \cdot (b_0 + b_1) - z_0 - z_2 \end{aligned}$$

Après le calcul des facteurs z_0 et z_2, le terme z_1 ne nécessite qu'une seule multiplication au lieu de deux avec l'algorithme classique, abaissant à trois le nombre de multiplications sur des entiers de β bits pour le calcul de c. L'algorithme de Karatsuba s'écrit donc :

$$c = a_0 b_0 + ((a_0 + a_1) \cdot (b_0 + b_1) - a_0 b_0 - a_1 b_1) \cdot 2^\beta + a_1 b_1 \cdot 2^{2\beta}$$

Afin de réduire davantage la complexité de la multiplication, l'algorithme de Karatsuba peut être appliqué aux sous-produits z_0, z_1 et z_2. Le nombre de fois où l'on applique l'algorithme de Karatsuba aux sous-produits est appelé le nombre de récursions.

Pour la version polynomiale, l'approche est similaire. On divise les coefficients des polynômes en deux parties. On pose :

$$A = A_L + A_H \cdot X^{\lceil n/2 \rceil}$$
$$B = B_L + B_H \cdot X^{\lceil n/2 \rceil}$$

A_L (resp. B_L) sont les coefficients de poids faible du polynôme A (resp. B), et A_H (resp. B_H) les coefficients de poids fort. L'algorithme de Karatsuba peut s'écrire :

$$C = A_L B_L + ((A_L + A_H) \cdot (B_L + B_H) - A_L B_L - A_H B_H) \cdot X^{\lceil n/2 \rceil} + A_H B_H \cdot X^{2\lceil n/2 \rceil}$$

5.2.3 Principes de l'architecture proposée

Nous avons étudié puis développé une plateforme permettant l'accélération des traitements du schéma de chiffrement FV. Pour cela, nous nous sommes appuyés sur une architecture comprenant un processeur généraliste et un accélérateur matériel. La taille des opérandes manipulés pouvant atteindre le million de bits, un FPGA de grande capacité est nécessaire pour mettre en oeuvre la solution proposée et réaliser nos expériences. La plateforme DE5-net de Terasic, offrant un FPGA Alera Stratix-V GX et une connectique PCIe, a été retenue.

L'arithmétique du schéma de chiffrement homomorphe FV utilisant des polynômes dans $\mathbb{Z}_q(X)/f(X)$, il est également nécessaire de réaliser des opérations de réduction modulaire polynomiale. L'addition et la soustraction de polynômes peuvent être efficacement implémentées de manière logicielle grâce à la programmation vectorielle. Cela permet en outre de calculer efficacement les pré- et post-récursions. Cependant, avec l'augmentation du nombre de récursions, l'approche purement logicielle atteint ses limites car il devient nécessaire de réaliser des opérations sur un nombre important de polynômes de degrés de plus en plus faible dont les opérandes sont éparpillés dans la mémoire. Cela engendre de fréquentes ruptures de pipeline réduisant les performances. Pour palier à cette limitation, l'approche proposée s'appuie sur un accélérateur matériel en charge des dernières récursions de l'algorithme de Karatsuba ainsi que toutes les multiplications entières des sous-produits.

La figure 4 présente la stratégie d'implémentation retenue de l'algorithme de multiplication basé sur Karatsuba. Pour cela, l'exemple de la multiplication polynomiale de polynômes avec 3072 coeffcients de 127 bits chacun est utilisé. Il est à noter que l'approche reste identique pour des tailles différentes. Pour chaque bloc présenté dans la figure 4, l'évolution de la taille et du nombre de sous-polynômes est indiqué. Dans cet exemple, 6 pré-récursions logicielles sont d'abord appliquées aux deux polynômes à multiplier A et B. Les sous-polynômes résultants sont ensuite regroupés dans un unique espace mémoire avant de les transmettre à l'accélérateur. Pendant ces transferts, l'accélérateur va procéder de manière pipelinée au calcul des dernières pré-récursions, à la multiplication des sous-produits et au calcul des premières post-récursions. À la suite de ces traitements, les données résultantes sont transmises à l'application logicielle, qui réalise les derniers post-traitements. Dans les travaux réalisés, cette approche a été étendue pour les opérations de multiplication homomorphe, de relinéarisation et de chiffrement pour le schéma FV.

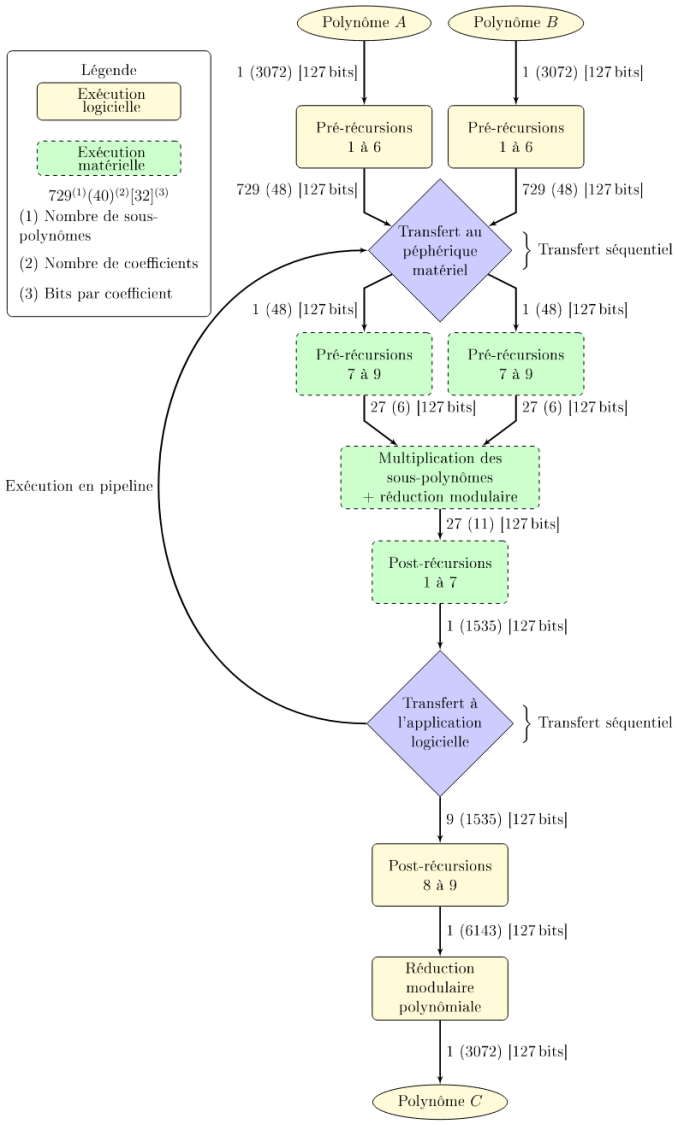


FIGURE 4 – Stratégie d'implémentation de l'accélération de la multiplication polynomiale avec Karatsuba dans une approche de co-conception

5.2.4 Principaux résultats

L'accélérateur matériel a été implémenté sur le FPGA Stratix V (GXEA7N2F45C2) de la plateforme DE5-net. La plateforme est interfacée via une interface PCIe à un PC exécutant Microsoft Windows 7 sur un processeur Intel Core i7-4790.

La table 7 présente la consommation de ressources de l'accélérateur développé. Les résultats sont présentés pour le cœur de l'accélérateur Karatsuba ainsi que pour l'interface pour le transfert des données depuis et vers les traitements logiciels. Le cœur de l'accélérateur Karatsuba est le module consommant la plus grande part des *arithmetic logic modules* (ALMs) requis. La moitié de ceux-ci sont utilisés comme élément mémoire afin d'assurer le stockage temporaire des données dans les différents étages de pipeline de l'accélérateur. La quantité de mémoire instanciée ($>$ 1 Mo) est nécessaire pour réaliser l'interfaçage entre l'accélérateur Karatsuba et l'interface PCIe utilisée.

La solution proposée est comparée, dans la table 8, avec une implémentation logicielle permettant l'utilisation du *batching* et utilisant la librairie NFLlib [Aguilar-Melchor 2016] s'appuyant sur la NTT et le système RNS (*Residue Number Systems*) favorisant la parallélisation des opéra-

TABLE 7 – Résultats d'implémentation pour l'accélérateur proposé

		Interface	Karatsuba
paramètres		$n = 2560$, $log_2 q = 135$	
ALM	logic	8 493	30 566
	memory	110	30 030
	total	8 603	60 596
registers		11 537	79 440
memory bits		9 162 104	25 164
DSPs		0	80
$fmax$		250 MHz (limitation du PCIe)	

TABLE 8 – Performance temporelle de l'accélérateur proposé

Temps d'exécution moyen (écart type) $n = 2560$, $log_2 q = 135$		
Chiffrement	Solution proposée	NFLlib
1	1 935 μs (220μs)	3 078 μs (98μs)
2	2 191 μs (138μs)	5 646 μs (148μs)
3	2 410 μs (176μs)	8 218 μs (191μs)
4	2 525 μs (184μs)	10 814 μs (237μs)

tions de multiplications. Les résultats montrent que l'accélérateur proposé est 1,5 fois plus rapide que la solution logicielle pour la réalisation de l'opération de produit-accumulation nécessaire lors de l'opération de chiffrement et 4 fois plus rapide pour 4 chiffrements réalisés en parallèle via la technique du *batching*.

Les contributions présentées dans cette section sont détaillées dans l'article ci-dessous publié en 2018 [Migliore 2018].

A High-Speed Accelerator for Homomorphic Encryption using the Karatsuba Algorithm

VINCENT MIGLIORE, CÉDRIC SEGUIN, MARIA MÉNDEZ REAL, VIANNEY LAPOTRE, ARNAUD TISSERAND, CAROLINE FONTAINE, and GUY GOGNIAT, Univ. Bretagne-Sud, UMR CNRS 6285, Lab-STICC, F-56100 Lorient, France
RUSSELL TESSIER, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA

Somewhat Homomorphic Encryption (SHE) schemes can be used to carry out operations on ciphered data. In a cloud computing scenario, personal information can be processed secretly, inferring a high level of confidentiality. The principle limitation of SHE is the size of ciphertext compared to the size of the message. This issue can be addressed by using a batching technique that "packs" several messages into one ciphertext. However, this method leads to important drawbacks in standard implementations. This paper presents a fast hardware/software co-design implementation of an encryption procedure using the Karatsuba algorithm. Our hardware accelerator is 1.5 times faster than the state of the art for 1 encryption and 4 times faster for 4 encryptions.

CCS Concepts: • **Security and privacy → Public key encryption**; **Hardware security implementation**; • **Hardware → Hardware accelerators**;

Additional Key Words and Pharses: Homomorphic Encryption, Karatsuba, hardware accelerator, FV

ACM Reference format:
Vincent Migliore, Cédric Seguin, Maria Méndez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine, Guy Gogniat, and Russell Tessier. 2017. A High-Speed Accelerator for Homomorphic Encryption using the Karatsuba Algorithm. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 138 (September 2017), 17 pages.
https://doi.org/10.1145/3126558

1 INTRODUCTION

Homomorphic Encryption (HE) is a recent promising tool in modern cryptography that supports operations on encrypted data. This property allows for the protection of private and sensitive data in a cloud computing scenario. Figure 1 provides a flowchart of a basic Homomorphic cloud service. Historically speaking, early cryptographic schemes presented partial homomorphic properties, for multiplication [1] and addition [2]. Only after the approaches in [3] and [4] were presented was it possible to support both types of operations at the same time. These schemes have been followed by many other related contributions.

This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2017 and appears as part of the ESWEEK-TECS special issue.
Authors' addresses: V. Migliore, C. Seguin, M. M. Real, V. Lapotre, A. Tisserand, and G. Gogniat, Université de Bretagne Sud, Rue de Saint-Maudé, 56100 Lorient; emails: {vincent.migliore, cedric.seguin, maria.mendez, vianney.lapotre, arnaud.tisserand, guy.gogniat}@univ-ubs.fr; C. Fontaine, Institut Mines-Telecom Atlantique, Campus de Brest, Technopôle Brest-Iroise, 29238 Brest; email: caroline.fontaine@imt-atlantique.fr; R. Tessier, University of Massachusetts, 134 Marston Hall, Amherst, MA 01003; email: tessier@umass.edu.

https://doi.org/10.1145/3126558

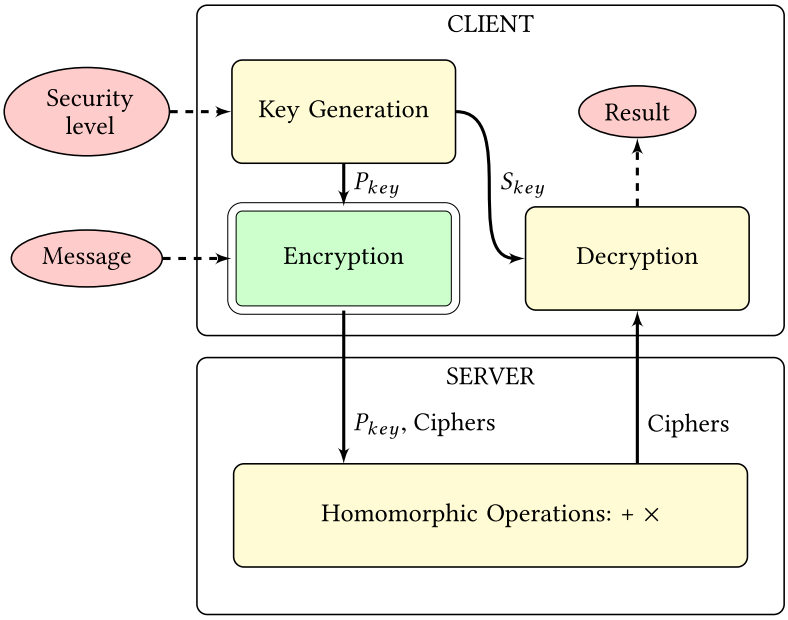


Fig. 1. Flow of an homomorphic cloud service.

Most promising Fully Homomorphic Encryption (FHE) schemes base their arithmetic on a ring of polynomials with integer coefficients [5], [6], [7], [8], [9], [10]. Each operation requires a double reduction: a modular reduction by an irreducible polynomial, typically required after each polynomial multiplication, and an integer reduction on each polynomial coefficient.

A key aspect of Homomorphic Encryption is the representation of messages. A message can be seen as a binary value, or an integer if it has more than one bit. If messages are represented as integers, only integer additions, subtractions and multiplications are possible. This list excludes standard operators like comparison. To enable such operations, it is necessary to switch to a binary message representation. The maximum value of a message is called the message space.

A second aspect of Homomorphic Encryption is the size of the encrypted data. Depending on the complexity of the cloud service, the cipher size can vary from a few KB to a few MB for 1 message (which can be binary). To address this penalty, two main solutions exist: transciphering and batching.With transciphering, data is sent to the server using standard symmetric encryption and then is decrypted on the server-side with homomorphic encryption. This operation requires a symmetric secret key, which is sent to the server encrypted with homomorphic encryption. This technique is not in the scope of this paper, but the reader can refer to [11] for further information. With batching, several messages are "packed" within one ciphertext using the Chinese Remainder Theorem (CRT) [12]. When homomorphic operations are computed on the server side, operations are executed for each message in parallel. Thus, the size of encrypted data per bit of information is reduced by the number of messages packed.

In practice, actual implementations of homomorphic encryption cannot perform batching efficiently due to the limitations of the standard algorithm used for such computation, i.e the NTT algorithm [13]. To perform a batching operation, the irreducible polynomial chosen for the FHE scheme must be factorizable in the message space. In particular, $x^n + 1$, the most efficient choice for NTT, is only factorizable for integer messages and not binary ones, and to our knowledge there is no efficient alternative. Without batching, NTT is a very powerful tool because once polynomials are converted to their NTT form, all computations are performed modulo $x^n + 1$ and thus, one can perform all required computations in this form.

In the batching case, the drawback is very significant. First, one needs to double the number of NTT points to avoid performing polynomial reduction during computations. Second, each

polynomial multiplication must perform one NTT, one component-wise multiplication, and then one inverted NTT. This process is in contrast to the no-batching case where polynomials remain in NTT form. On the client side, because computing capacity is limited, this drawback can become very costly. As a consequence, the well known SEAL library only implements batching for non binary messages, which reduces the interest of the implementation for many algorithms. In practice, few approaches use hardware to target homomorphic encryption with batching for binary messages due to the incompatibility with NTT. To our knowledge, only [14] and [15] provide complete accelerators with batching, although the former focuses on the server side only and the latter only targets complex homomorphic algorithms. In this paper, we provide the first batching compliant implementation of homomorphic encryption on the client side using the Karatsuba algorithm [16], and compare our results to the latest homomorphic libraries. The implementation greatly extends the work in [15] with important modifications to adapt the computation to the encryption both in terms of software and hardware. The main contributions of this work are as follows:

- Encryption step acceleration using a hardware/software co-design approach that leverages the Karatsuba algorithm (up to 4 ciphers in parallel).
- High performance software computations using vector programming (AVX2/SSE4.2 and NEON).

Compared to the server-side accelerator [15] in which parallel operations could not be performed due to the complexity of the homomorphic multiplication, we have exploited the polynomial arithmetic simplicity of the encryption operation to parallelize our design. Thus, the main challenge of this work was simultaneously dealing with a larger hardware design and much larger transfers between hardware and software. This limitation has been addressed by exploiting the structure of polynomials in Homomorphic Encryption.

This paper is organized as follows. Section 2 provides notation and basic mathematical knowledge about encryption and the Karatsuba algorithm. Section 3 describes the hardware and software accelerator architecture. Section 4 presents implementation results and compares them with a state of the art software implementation. Section 5 summarizes and concludes the paper.

2 THEORETICAL BACKGROUND

2.1 Notation

In the following, a polynomial is represented in uppercase and its coefficients in lowercase. For polynomial A, a_i represents its i^{th} coefficient. A vector of polynomials is noted in bold. For vector $\mathbf{A}$, $\mathbf{A}[i]$ is the i^{th} polynomial of the vector. For set R and polynomial A, $A \leftarrow U_R$ represents a uniformly sampled polynomial in R, $A \leftarrow \chi_\sigma$ is a polynomial sampled in a discrete Gaussian distribution with standard deviation σ and B_R a very narrow discrete Gaussian distribution in which polynomials have binary coefficients. For coefficient a_i of polynomial A, $a_{i,(j..k)}$ corresponds to the binary string extraction of a_i between bits j and k. This notation is extended to polynomial A where $A_{(j..k)}$ is the sub-polynomial in which the binary string extraction is applied to each coefficient. A modular reduction by an integer q is $[\cdot]_q$. For integer a, $\lfloor a \rfloor$, $\lceil a \rceil$ and $\lfloor a \rceil$ operators are floor, ceil and nearest rounding operations, respectively. This notation is extended to polynomials by applying the operation on each coefficient. For vectors $\mathbf{A}$ and $\mathbf{B}$, $\langle \mathbf{A}, \mathbf{B} \rangle$ represents $\sum \mathbf{A}[i]\mathbf{B}[i]$. In the following, polynomials have coefficients in $\mathbb{Z}_q$, i.e. integer coefficients in $[0, q[$.

2.2 Ciphering

This paper focuses on the encryption operation for the Ring-Learning With Error (R-LWE) [17] based schemes, and in particular the Fan Vercauteren (FV) [7] scheme. The basic idea of R-LWE is to

hide a secret by using a noisy distribution. For a secret polynomial $S \leftarrow B_R$ and noisy polynomials $A \leftarrow U_R$ and $E \leftarrow \chi_R$, a R-LWE sample is the couple $(-A \cdot S + E, A)$. In the case of FV, $S_{key} = S$ is the secret key and $\mathbf{P_{key}} = (-A \cdot S + E, A)$ is the public key. For an integer t such as a message $m \in [0, t[$ (integer message), the encryption operation is performed as follows:

$$\mathbf{C} = \left(\left[\Delta m + \mathbf{P_{key}}[1]U + E_1 \right]_q, \left[\mathbf{P_{key}}[2]U + E_2 \right]_q \right) \tag{1}$$

with $U \leftarrow B_R$, $(E_1, E_2) \leftarrow \chi_R^2$ and $\Delta = \lfloor q/t \rfloor$. If the noise term $EU + E_1 + S_{key}E_2$ is below $\Delta/2$, the message can be decrypted without error. In the following, we set $t = 2$ which is a common choice in the literature. In particular, it allows a wide range of operations (such as comparison) instead of just integer addition, subtraction and multiplication (i.e. when $t > 2$).

To accelerate this operation, we use a software/hardware co-design implementation of a polynomial multiplication algorithm which is based on the Karatsuba algorithm. For ciphering, Karatsuba efficiently performs $\mathbf{P_{key}}[1]U$ and $\mathbf{P_{key}}[2]U$ operations. In addition, high-speed binary polynomial generation and a discrete Gaussian sampler are required to generate U, E_1 and E_2. However, we decided to do not include these primitives in the scope of this work as we believe a more mature background is required.

2.3 The Batching Technique

The arithmetic of R-LWE schemes must perform polynomial operations modulo an irreducible polynomial in $\mathbb{Z}_q$ (usually chosen in the literature as a cyclotomic polynomial for security concerns). Some cyclotomic polynomials have an additional property, they are reducible in $\mathbb{Z}_2$. If each factor is unique and has a multiplicative order of 1, then the batching technique is possible. Formally, for a vector of k messages $\boldsymbol{m}$ and a given irreducible polynomial Φ in $\mathbb{Z}_q$ compatible with batching, the batching polynomial M can be expressed as:

$$M = \sum_{i=1}^{k} \boldsymbol{m}_i \cdot \boldsymbol{S}_i \cdot \boldsymbol{\Phi}_i \mod \Phi, \text{ with } \begin{cases} \Phi \equiv \prod_{i=1}^{k} \boldsymbol{\varphi}_i \mod 2 \\ \boldsymbol{\Phi}_i = \dfrac{\Phi}{\boldsymbol{\varphi}_i} \\ \boldsymbol{S}_i \equiv \boldsymbol{\Phi}_i^{-1} \mod \varphi_i \end{cases} \tag{2}$$

To recover the i^{th} message, it suffices to perform

$$\boldsymbol{m}_i \equiv M \mod \varphi_i \tag{3}$$

$\boldsymbol{m}_i$ is called the residue polynomial. Then, for two batching polynomials M_a and M_b, polynomial additions, subtractions and multiplications, perform the same operation between residue polynomials in parallel.

As stated in Section 1, the best choice for NTT is the cyclotomic polynomial $x^n + 1$. With such parameters, NTT can be adapted to compute polynomial modular reduction during computations (called Negative Wrapped Convolution). However, the factorization of $x^n + 1$ in $\mathbb{Z}_2$ is $(x + 1)^n$. So this polynomial is not compatible with batching due to the unique factor and the multiplicative order. NTT Positive Wrapped Convolution is the current alternative. It performs polynomial arithmetic modulo $x^n - 1$ during computations. However, $x^n - 1$ is not compatible with homomorphic encryption because it is clearly reducible (1 is an obvious root). This greatly penalizes NTT, which requires several adaptations. First, the number of points of the NTT algorithm must be twice as large versus the Negative Wrapped Convolution case to not perform polynomial reduction. This issue is quite critical when the degree is slightly higher than a power of two. For

example, for degree-3000 polynomial multiplication, the required NTT has $2 \times 4096 = 8192$ points. Second, modular reduction by a cyclotomic polynomial Φ must be carried out. This implies a need to perform an inverted-NTT, a modular reduction, and then an NTT.

These limitations show that NTT has important issues dealing with batching. This issue motivates our architecture based on the concurrent polynomial multiplication algorithm called Karatsuba. Karatsuba does not suffer from batching limitations and it is possible to adapt the algorithm to various polynomial sizes.

2.4 Karatsuba Algorithm

The Karatsuba algorithm is an improvement on the standard polynomial multiplication algorithm which reduces the number of sub-products. In SHE, polynomials have the same number of coefficients, and our setup always provides an even number of coefficients. Thus, in the following, we only discuss the Karatsuba algorithm with these constraints. Input polynomials A and B of degree $n-1$ are split into two parts of equivalent size, $\frac{n}{2}$ coefficients. Let A_H and A_L be two polynomials composed of the coefficients of the highest degree of A and the lowest degree of A, respectively. B_H and B_L are constructed using the same approach. Input polynomials can now be expressed as $A = A_L + A_H x^{n/2}$ and $B = B_L + B_H x^{n/2}$.

When A and B are multiplied using the standard approach, the resulting decomposition is given by:

$$\begin{aligned} A \times B &= (A_L + A_H x^{n/2})(B_L + B_H x^{n/2}) \\ &= A_L B_L + (A_L B_H + A_H B_L) x^{n/2} + A_H B_H x^n \end{aligned} \tag{4}$$

Karatsuba optimization exploits the fact that the middle factor $(A_L B_H + A_H B_L)$ can be cleverly computed as $(A_L + A_H)(B_L + B_H) - A_L B_L - A_H B_H$. $A_L B_L$ and $A_H B_H$ are already computed and do not require additional multiplications.

In total, Karatsuba requires 3 sub-polynomial multiplications instead of 4, at a cost of two pre-computations, $(A_H + A_L)$ and $(B_H + B_L)$, and two post-computations for the reconstruction of the middle factor. These pre- and post-computations only require additions and subtractions. To further reduce the complexity of the polynomial multiplication, one can apply recursively the Karatsuba algorithm to each sub-polynomial multiplication, $A_L B_L$, $A_H B_H$ and $(A_H + A_L)(B_H + B_L)$. The number of times that the Karatsuba algorithm is applied is called the number of Karatsuba recursions. After several Karatsuba recursions, one has to perform many low degree polynomial multiplications instead of a large polynomial multiplication. This recursiveness allows computation sharing between software and hardware. For example, several recursions can be performed in software and the remaining ones in hardware.

Because each Karatsuba recursion halves the size of sub-polynomials, Karatsuba can achieve polynomial multiplication of degree $2^r(p+1) - 1$, where r is the number of Karatsuba recursions and p the degree of the smallest sub-polynomial.

3 ACCELERATOR ARCHITECTURE

3.1 High-Level Overview

We based our architecture on the design in [15] which uses Karatsuba algorithm to accelerate server-side operations. This work uses the same Homomorphic Encryption setup (e.g. degree-2559 polynomials with 125-bit coefficients) to speed-up client-side operations, in particular the ciphering. Figure 2 presents the flow of the proposed accelerator that supports 4 parallel encryptions. The accelerator operates as follows: Six Karatsuba pre-recursions are computed in software and three are performed in hardware. After software pre-computations for each input polynomial,

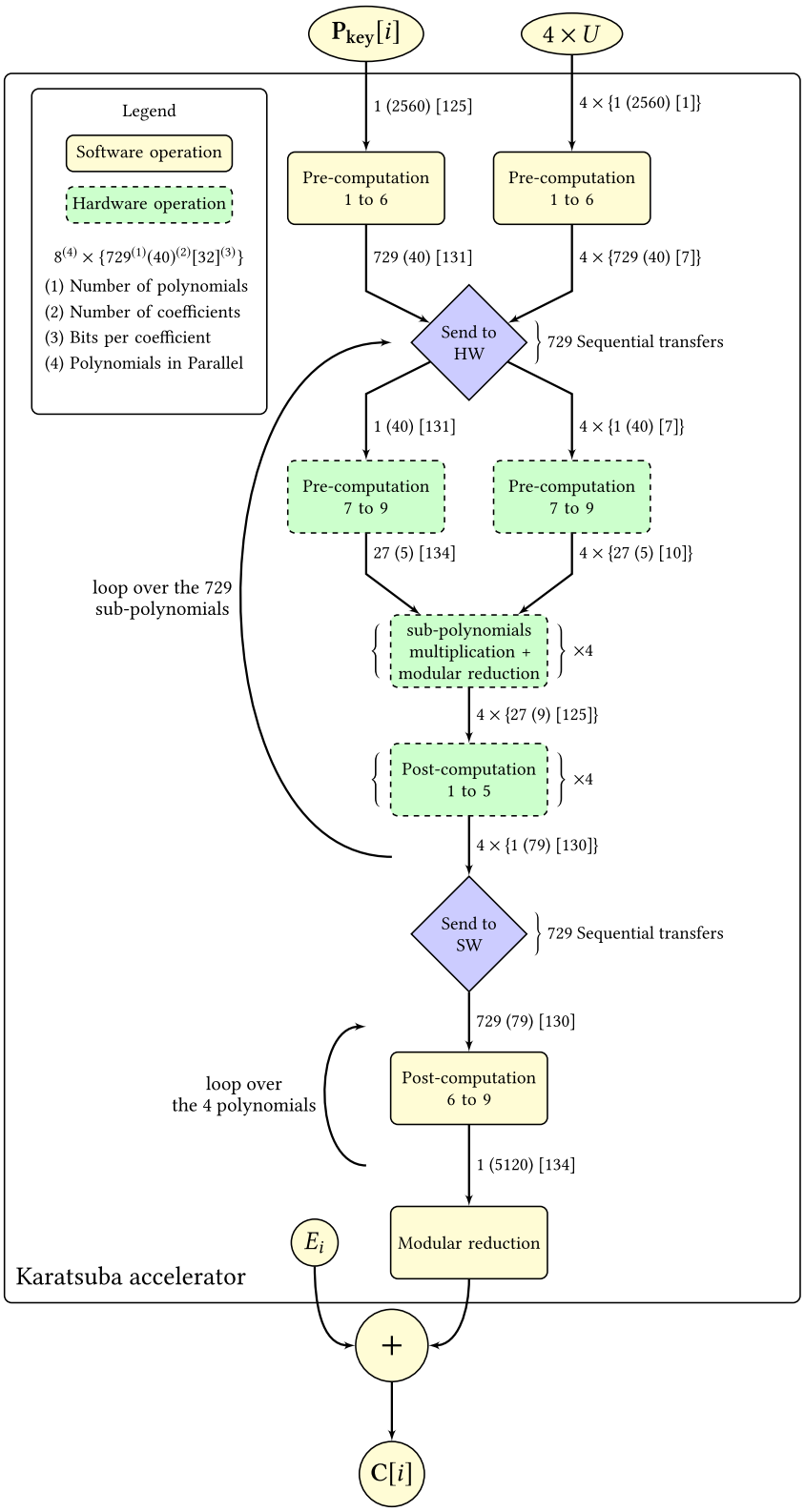


Fig. 2. Flow of the encryption operation in our architecture, where $i \in \{1, 2\}$ represents the i^{th} member of the ciphertext C. Values $\mathbf{P_{key}}[i]$, U, and E_i are, respectively, the public key, a binary sampled polynomial and a Gaussian sampled polynomial.

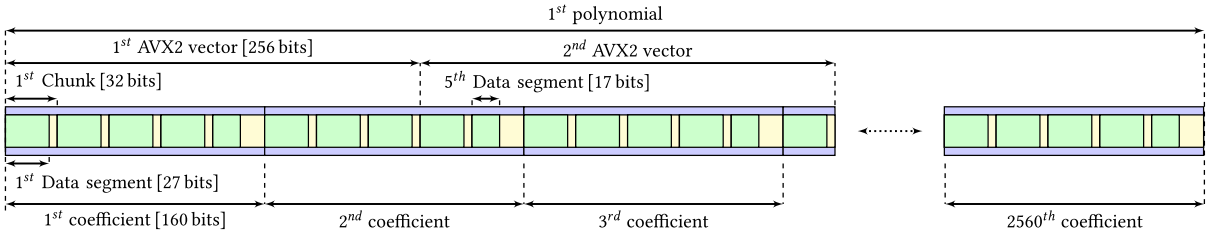


Fig. 3. Software representation of one polynomial of the public key.

729 sub-polynomials with 40 coefficients are generated and sent sequentially to the hardware. The hardware is fully pipelined and operations on sub-polynomials are executed as soon as polynomials arrive. At the end of Karatsuba pre-computations, 19,683 degree-4 polynomials are generated on each input polynomial that must be multiplied term by term. These multiplications are computed in hardware by four parallel polynomial multiplier units using the standard polynomial multiplication algorithm. Because the accelerator computes up to four encryptions in parallel, four post-computation units are implemented in parallel in hardware. These post-computations are computed sequentially in software, although multi-threading can possibly be used. The distribution of Karatsuba pre- and post-recursions between hardware and software is a key element of our architecture. Further details are provided in Section 3.3.5.

3.2 Software Implementation

Our Karatsuba software design is implemented using contemporary vector programming (AVX2, SSE4.2, NEON, . . .). For simplicity, we describe our work for the AVX2 instruction set, but the approach remains valid for SSE4.2 and NEON with the exception that their vectors have smaller length. AVX2 Single Instruction Multiple Data (SIMD) instructions are performed on 256-bit vectors. The vector can be seen as 4 doubles (64-bit operands), 8 floats (32-bit operands) or 8 integers (32-bit operands). For each elementary operation, the computation is performed on each element of the vector in parallel. AVX2 supports additions, subtractions, multiplications, maskings and various methods to speed-up specific algorithms.

3.2.1 Representation of Polynomials in Memory. An efficient representation of polynomials in memory has been made to enhance the efficiency of vector programming with Karatsuba. As mentioned in Section 2.2, Karatsuba multiplies the public key $\mathbf{P_{key}}$ with a randomly-generated binary polynomial. Thus, two different kinds of storage are required: a full size polynomial with 125 bits per coefficient for the public key, and a small polynomial with 1 bit per coefficient for the binary polynomial.

Figures 3 and 4 provide data representations of a full size polynomial and a binary polynomial, respectively. For the full size polynomial, coefficients are split into five 32-bit chunks. Because AVX2 operations do not support addition with carry, guard bits are required for carry propagation between operations. In our case, this choice was quite simple because with 125-bit coefficients, it is not possible to have less than five chunks to be able to have at least one guard bit per chunk. This setup has the benefit of providing multiple guard bits per chunk, which allows for the computation of successive operations before carry propagation. Section 3.2.2 provides a further explanation on the impact of guard bits. For binary polynomials, only 1 bit per coefficient is needed, implying an inefficient use of memory and unnecessary computation overhead if the previous memory scheme is followed. Figure 4 provides an optimized representation with 20 times less memory consumption. First, the number of chunks per coefficient is reduced to one. Second, multiple coefficients are stored per chunk. Because our Karatsuba implementation has six recursions in software, at least

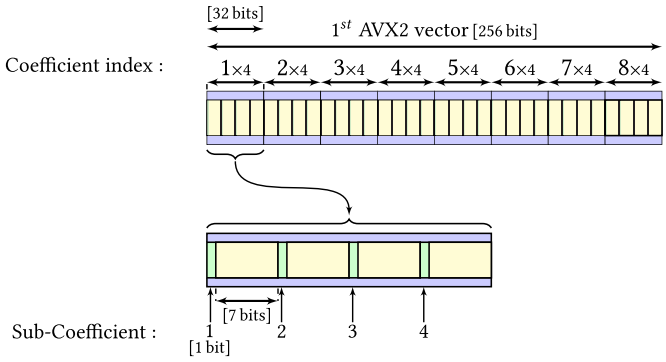


Fig. 4. Software representation of binary polynomials.

six guard bits are required. Thus, one 32-bit chunk can store up to four coefficients. For efficiency considerations, we decided to store four polynomials using 2,560 chunks instead of one using 640 chunks.

3.2.2 Elementary Polynomial Arithmetic. Karatsuba pre- and post-computations are quite simple and only require polynomial additions and subtractions. That is why, our software implementation focuses on polynomial addition and subtraction. As a reminder, our setup provides five guard bits per chunk, and fifteen guard bits at the coefficient level. For addition, operations can be easily implemented. With five guard bits per chunk, five additions at the chunk level are allowed before an overlap. This effect is a consequence of the fact that an addition can increase the result by one bit in the worst case. Then, a coefficient reconstruction is required to restore the guard bits. This operation consists of taking the guard bits of one chunk, and adding them to the next one like a standard carry propagation. This operation, apart from restoring the guard bits, has the consequence of consuming the guard bits of the last chunk of each coefficient. As the addition consumes one guard bit per chunk, one guard bit at the coefficient level is consumed per addition. Therefore, our data representation supports fifteen successive polynomial additions before requiring a modular reduction of each coefficient.

The subtraction operation uses standard two's complement arithmetic. For integer subtraction of two integers A and B with $\log_2 q$ bits, two's complement subtraction can be expressed as :

$$A + \overline{B} + 1 = A - B \tag{5}$$

with $\overline{B}$ the binary inverse of B. In the standard case, carry propagation is performed during the subtraction computation. In our case, due to the guard bits, we cannot invert chunks directly. If we note t_{a_i} the guard bits of a chunk a_i and m_{a_i} the other bits, we must in fact compute the inverse of $t_{a_i} + m_{a_{i+1}}$. This operation creates data dependencies between chunks, breaking the parallelism of computations. This can be easily addressed using the following property:

$$\begin{aligned} \overline{t_{a_i} + m_{a_{i+1}}} &= -t_{a_i} - m_{a_{i+1}} - 1 \\ &= -t_{a_i} - 1 - m_{a_{i+1}} - 1 + 1 \\ &= \overline{t_{a_i}} + \overline{m_{a_{i+1}}} + 1 \end{aligned} \tag{6}$$

We just keep in mind that now the leading bit of a given chunk provides the sign. Thus, it reduces by 1 bit the number of guard bits during subtractions compared to additions.

The asymmetry between AVX2 vector length, polynomial length, and coefficient length must also be considered. An AVX2 vector has 256 bits which covers eight chunks. At the coefficient level, the fact that a AVX2 vector is longer than a coefficient is not a problem. Because coefficients

are reconstructed later, a point-wise addition on the chunks is sufficient. At the polynomial level, the situation is not as simple. The computation process, and especially Karatsuba computations, leads to the creation of polynomials of various sizes. If the polynomial size is not a multiple of the AVX2 vector, a data overlap is possible. In this case, a masking operation on the latest operation is required to prevent operations and results outside of the valid data range.

3.2.3 Polynomial Modular Reduction. Polynomial modular reduction is performed after each polynomial multiplication. Without batching, this operation is quite simple. For example, with the cyclotomic polynomial $x^n + 1$, the reduction of a polynomial $A = A_L + A_H \cdot x^n$ is $A_L - A_H$. With batching, the cyclotomic polynomial may have numerous non-zero coefficients (Hamming weight), leading to a possible complex reduction. The standard algorithm for such an operation is the Barrett reduction [18]. However, this algorithm requires two polynomial multiplications and one polynomial subtraction which is undesirable for performance. To address the reduction in our design, we have made an exhaustive search on cyclotomic polynomials to extract good candidates compatible with batching, i.e. with the smallest Hamming weight. Then, we have exploited the structure of these polynomials to perform polynomial modular reduction as an addition/subtraction of sub-polynomials. A degree-n cyclotomic polynomial can be written as:

$$\Phi = \sum_{i=0}^{m} \alpha_i \cdot X^{\beta \cdot i}, \text{ with } \begin{cases} \alpha \in \{-1, 0, 1\} \\ (m, \beta) \in \mathbb{Z} \text{ such as } n = m \cdot \beta \end{cases} \tag{7}$$

Because α_i is an integer and Φ has been chosen to maximize β, the underlying polynomial has numerous empty coefficients which greatly simplifies the polynomial reduction.

The polynomial reduction algorithm is performed by solving a system of equations. We note A, a polynomial to be reduced by a cyclotomic polynomial Φ, B the quotient polynomial and R the residue. First, we split polynomials A, B and R into several degree-β polynomials.

$$A = \sum_{i=0}^{2 \cdot m-1} A_i \cdot X^{\beta i}, B = \sum_{i=0}^{m-1} B_i \cdot X^{\beta i}, R = \sum_{i=0}^{m-1} R_i \cdot X^{\beta i} \tag{8}$$

Second, we extract the equations system by exploiting the relationship between A, B and R:

$$A = B \cdot \Phi + R \tag{9}$$

Because $\deg R < n$, we have the following system:

$$\begin{cases} \forall i \in \{m, 2m\}, A_i = \sum_{j=0}^{i} B_j \cdot \alpha_{i-j} & \text{(a)} \\ \forall i \in \{0, m-1\}, A_i = \sum_{j=0}^{i} B_j \cdot \alpha_{i-j} + R_i & \text{(b)} \end{cases} \tag{10}$$

With Equation (10(a)), we can calculate the different B_i, and Equation (10(b)) determines the residue polynomial R. We have implemented a script to automatically determine the different R_i. For degree-2560 polynomials with 125-bit coefficients (22 batches), our software library performs the polynomial reduction in 114 μs on average (1000 runs). This value is competitive with Barrett reduction, because a simple polynomial multiplication using NFLlib for the same parameters costs 1.7 ms on average. To improve the performance of the polynomial modular reduction, we first compute sub-polynomial additions to maximize the number of successive operations before reconstruction. Then subtractions are performed with the limitation explained in Section 3.2.2.

3.2.4 High-speed Batching Implementation. One of the key elements for batching to be a good alternative to the standard approach is the batching cost. In Equation (2), for k batches, possibly $2 \cdot k$ polynomial multiplications, k polynomial additions and 1 polynomial reduction are required. In our architecture, we need to avoid polynomial multiplications as much as possible for efficiency. We performed several optimizations to greatly reduce the complexity of this step. First, $S_i \cdot \Phi_i$ in Equation (2) can be precomputed as they are constant for a given cyclotomic polynomial Φ. Second, messages are binary so the product $m_i \cdot S_i \cdot \Phi_i$ can be replaced by a simple test. Third, $\deg m_i = 0$, $\deg S_i = \deg \varphi_i - 1$ and $\deg \Phi_i = \deg \Phi - \deg \phi_i$, so $\deg m_i \cdot S_i \cdot \Phi_i = \deg \Phi - 1$. Thus the final polynomial reduction can be avoided. Finally, the complete process is reduced to k conditional polynomial additions, which can be quickly implemented.

3.2.5 Karatsuba Pre- and Post-computation Details. Karatsuba pre- and post-computations have been implemented with a recursive algorithm which follows the approach presented in [15]. The pre-computation is quite easy to implement because operations are polynomials additions only. It is only necessary to reconstruct each coefficient after at most five successive polynomial additions, as explained in Section 3.2.2. When this operation is performed it is optimized to reduce performance overhead. The Karatsuba algorithm is composed of several successive recursions. At the i^{th} recursion, sub-polynomials are the results of at most i sub-polynomial additions. We experimented with several approaches to determine the best moment to implement the reconstruction. In our case, the best results were achieved when the reconstruction is implemented during the first Karatsuba recursion. A simple reasoning leads to this result. Between each Karatsuba recursion, the number of sub-polynomials increases but each elementary sub-polynomial has lower coefficients. However, each recursion increases the total size of sub-polynomials by 1.5. Thus, the earlier the reconstruction is performed, the smaller the amount of data that must be considered. The last recursion also requires reconstruction to support compatibility with the hardware accelerator. A minor modification to the hardware that performs this last reconstruction results in an accelerator performance improvement. Because the AVX2 vector performs addition on eight chunks in parallel, it is important to have a chunk count that is a multiple of eight to avoid masking operations. For a polynomial of 2,560 coefficients, this criterion is met during pre-computation since the smallest sub-polynomial addition has 80 coefficients after the fifth recursion. To prevent masking, four binary polynomials are stored in 2,560 chunks, as explained in Figure 3.

More issues are apparent for post-computation than for pre-computation. First, the number of coefficients is not a multiple of eight, so operations require masking. Second, data is not always aligned on 32-byte boundaries, so computation is penalized during non-aligned data loads. Third, post-computation requires successive polynomial subtractions which are subject to the limitations explained in Section 3.2.2. The final recursion of the pre-recursion and the first post-computation in software have been adapted to automatically deal with sub-polynomials in such a way that data is compatible with the PCI-E driver. The main benefit of this approach is avoiding data type conversion as much as possible.

3.3 Hardware Implementation

3.3.1 Architecture Overview. Software and hardware components in our system communicate via the PCI-E bus. The hardware accelerator is implemented on an FPGA (further details are provided in Section 4). A RIFFA [19] interface implemented for PCI-E Gen 3 with four lanes is used. On the FPGA side, the interface provides data bursts of 128 bits at 250 MHz. Figure 5 shows the hardware accelerator architecture. Computations are pipelined so computation operations are executed during transfers. The pre-computation units perform the remaining pre-computations and consist of three smaller units in cascade (one per recursion). For a sub-polynomial A, one

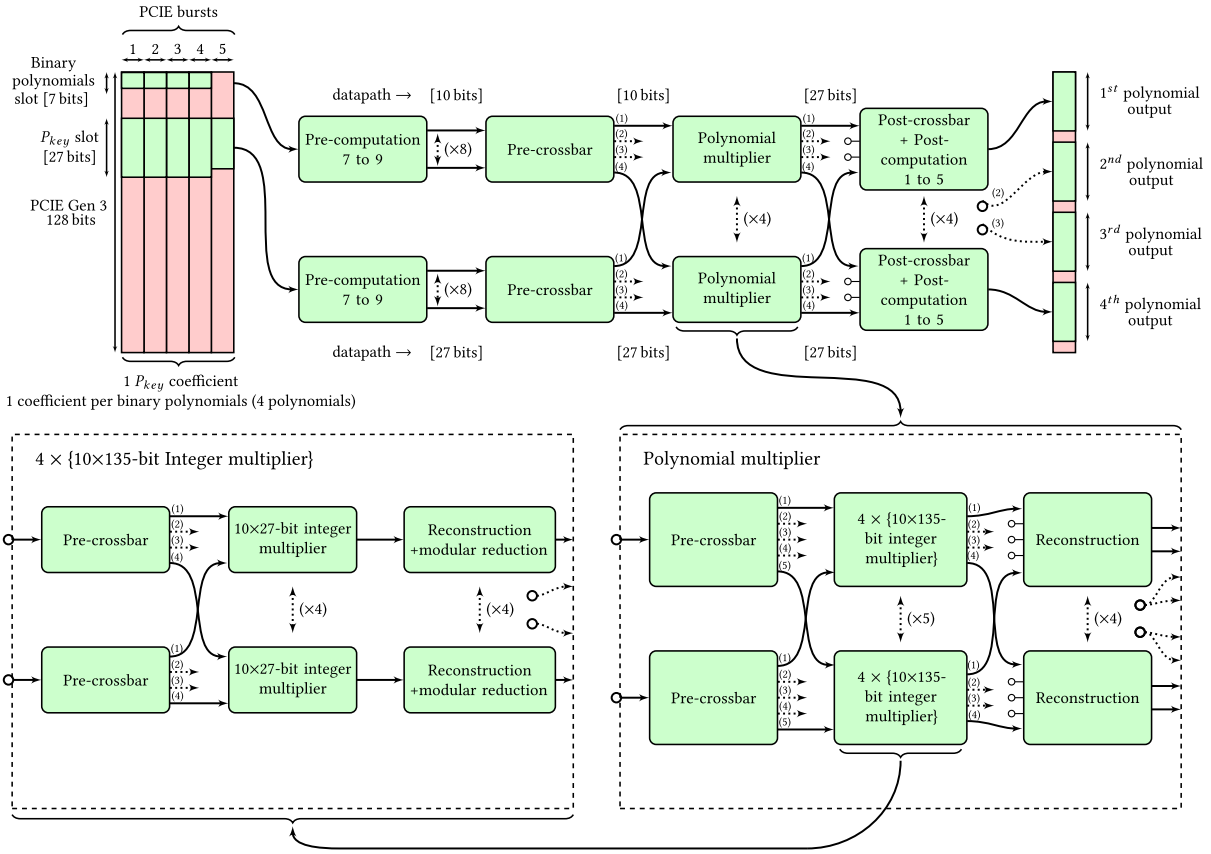


Fig. 5. Hardware accelerator architecture.

smaller unit produces sub-polynomials A_L, A_H and $A_L + A_H$ (according to notation provided in Section 2.4), and pushes these polynomials to the following unit to perform one pre-computation operation on A_L, A_H and $A_L + A_H$, respectively. After three recursions, the design has eight lines of sub-polynomials but many lanes are not fully fed by sub-polynomials. Thus a crossbar is implemented to better schedule sub-polynomials and reduce the number of sub-polynomial lanes. A sub-polynomial multiplier is implemented to multiply the different sub-polynomials with the standard polynomial algorithm. The post-crossbar and the post-computation units are represented in the same box because they are implemented successively but at the recursion level to reduce the storage overhead.

Many improvements have been made to the preliminary work in [15] to support encryption operations. First, the pre-computation was adapted to efficiently pre-compute four binary polynomials in parallel during public key pre-computation. Second, the polynomial multiplier core was modified to support the multiplication of four polynomials with unbalanced coefficient sizes (10×135-bit integer multipliers). Third, five post-computations were implemented in hardware instead of three to reduce bandwidth. These new aspects of the architecture are discussed in the following sections of the paper.

3.3.2 Adaptation of the Pre-computation. Two lane types are supported for the computations: one lane for the public key and a second for the binary polynomials. For the public key, coefficients are split into five 27-bit chunks. Elementary coefficient addition/subtraction is performed in five steps, which corresponds to a simple chunk-wise addition/subtraction with carry propagation. This setup is sufficient to store each coefficient and matches the software representation of

polynomials, simplifying the connection with software. The binary polynomials are much smaller than standard ones and only 32 bits are required to send one coefficient of the four binary polynomials. To maximize resource utilization, one coefficient of one binary polynomial is stored per PCI-E burst. This approach allows for the pre-computation of five binary polynomials during P_{key} pre-computation. In practice, processing is only performed in four out of the five available slots because the software only pre-computes four binary polynomials.

At the beginning of the pre-computation process, each binary polynomial coefficient has at most 7-bit width. Because three pre-computations are performed in hardware, in the worst case, output coefficients of the pre-computation unit have 10-bit widths. The addition with carry units in the previous pre-computation unit can be replaced by a simple adder and the datapath can be adjusted to 10 bits to avoid carry propagation.

3.3.3 Adaptation of the Polynomial Multiplier. The polynomial multiplier performs sub-polynomial multiplication of polynomials generated after the pre-computation process. These sub-polynomials have a low degree, four in our case, and are implemented with the standard multiplier algorithm. For efficiency, the multiplier is fully parallelized, and as shown in Figure 5, requires five integer multiplier units. These integer multiplier units have been designed to support four 10×135-bit integer multipliers in parallel to support four encryptions. Because binary polynomial operations are scheduled to improve resource utilization, the polynomial multiplication schedule has been adapted. Both polynomial multiplier and 4×{10×135-bit Integer Multiplier} units follow the same approach: A pre-crossbar to schedule incoming data, multiplier units, and a reconstruction unit. Figures 6 and 7 provide the schedule of the polynomial multiplier and internal 4×{10×135-bit integer multiplier} units. The block of data represented in gray is an elementary block of data processed by the internal integer multipliers. From a high level point of view, the polynomial multiplier pre-crossbar is responsible for supporting the convolution operation as the integer multiplier pre-crossbar separates the coefficients of the four binary polynomials into four different lanes. In Figure 7, it is apparent that five lanes of binary polynomials are possible. However, to be compliant with our software architecture, we only use four lanes instead of five.

3.3.4 Adaptation of Post-Computation Operations. Due to limited PCI-E bandwidth, it was necessary to implement two additional post-computations in hardware beyond the standard case. In the standard case of three pre- and post-computations, input polynomials have 40 coefficients with a 131-bit width and output polynomials have 79 coefficients with a 128-bit width due to integer modular reduction. Thus, twice the bandwidth is needed for the output compared to the input. Because chunks have 27 bits, for four polynomial multiplications, $2 \times 4 \times 27 = 216$ bits are needed for the output, which is larger than the 128 bits provided by RIFFA. Depending on the number of post-computations, it is possible to have fewer output coefficients than the number of required input coefficients. This assertion becomes true when implementing the two additional post-computations in hardware in our case. Output polynomials now have 319 coefficients, and the number of input polynomials for two post-recursions is nine, leading to 360 input coefficients. This architectural modification has both pros and cons. Implementing additional post-computations in hardware speeds-up the software post-computation process and reduces the size of the transfer between the FPGA and the software. However, one needs to ensure that nine successive input polynomials can be sent to the hardware accelerator without significant latency between them. As will be explained in the next section, RIFFA input buffer management must be carefully implemented.

3.3.5 Selection of the Distribution Ratio between Hardware and Software for Karatsuba Recursions. The distribution ratio of the Karatsuba recursions between hardware and software is a critical design choice. Although software can implement Karatsuba pre- and post-recursions,

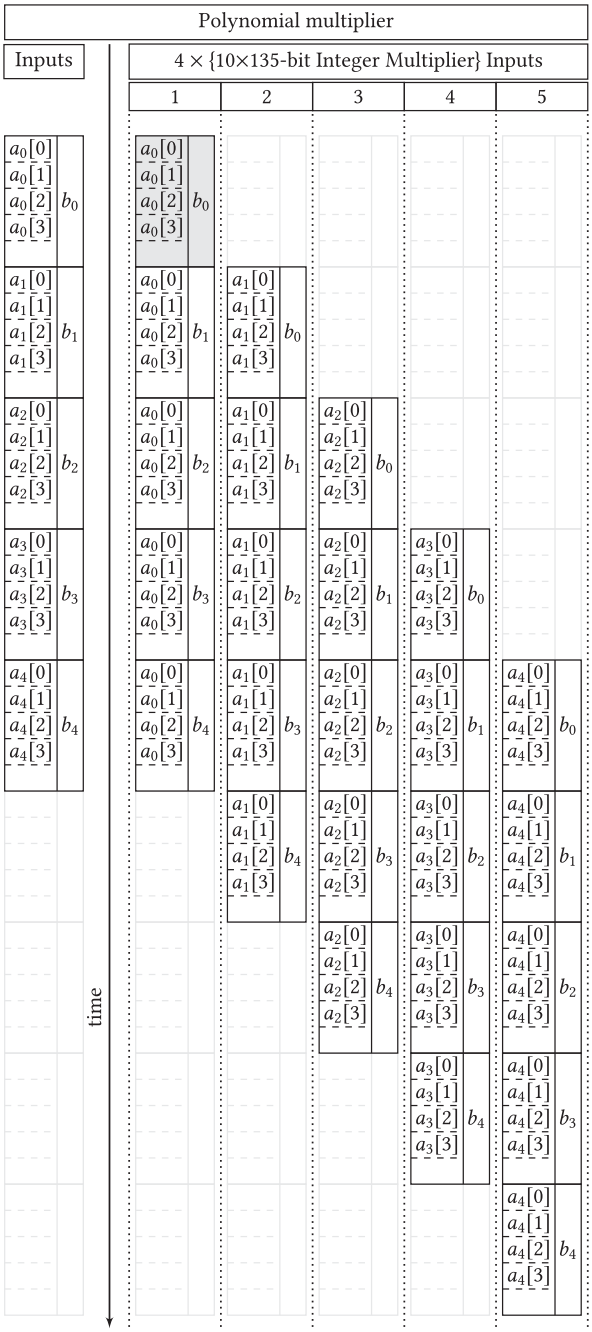


Fig. 6. Polynomial multiplier schedule. $a_i[j]$ represents the i^{th} coefficient of the j^{th} binary polynomial, and b_i the i^{th} coefficient of the public key. The gray block represents an element scheduled by the 4×{10×135-bit Integer Multiplier}.

implementing numerous recursions in software has several limitations. First, software recursions increase the transfer size through the PCI-E (as a reminder, each recursion increases the size of the transfer by 1.5x). Since the software computation time increases, the overall computation time increases as well. Second, post-computations in software are costly compared to pre-computation. Our experiments show that post-computation can be eight times slower than pre-computation. Moreover, for four parallel encryptions, post-computation must be performed four times in software.

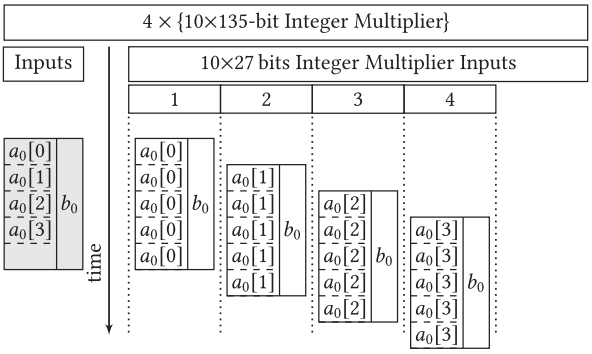


Fig. 7. 4×{10×135-bit Integer Multiplier} schedule. $a_i[j]$ represents the i^{th} coefficient of the j^{th} binary polynomial, and b_i the i^{th} coefficient of the public key.

Implementing additional post-recursions in hardware (i.e. more than the number of pre-recursions) is not a complex task because additional post-crossbars are unneeded. The main issue is the implementation of the pre-crossbar. For fewer than three pre-recursions, there is not enough parallelism to efficiently feed hardware multipliers (multipliers are unused 25% of the time). For more than three, parallel implementation requires a complex sub-polynomial schedule which can penalize the maximum design frequency. As a result, three pre-recursions are performed in hardware to limit pre-crossbar complexity. For the post-computation, only five post-recursions are performed due to PCI-E bandwidth limitations. This implementation greatly improves upon the software performance by reducing computation time.

3.3.6 Limitations of the PCI-E Interface. As mentioned earlier, a RIFFA 2.1 interface is used to make the connection between software and hardware components. The transfer bandwidth mostly depends on the size of the transfer [19]. For our system, instead of initiating one transfer per set of sub-polynomials, a large transfer with all pre-computed polynomials is initiated. As explained in the discussion in Section 3.3.1 regarding PCI-E bursts, a transfer requires the transmission of 729 degree-39 sub-polynomials, about 2.2 MB of data. With such a transfer, RIFFA should achieve a bandwidth of 3,000 MB/s which corresponds to a transfer time of 741 μs. In practice, we achieved a bandwidth of 2,000 MB, since our hardware accelerator shares the PCI-E bus with several components.

This limitation leads to two consequences. First, although the total hardware computation time (including transfers) is lengthened, the Karatsuba hardware computation can partially compensate since it can be performed during transfers. However, the transfer latency impact is not negligible. Compared to the best possible case of 583 μs to perform all hardware computations, performance is slowed down by 47%. Second, two extra components were designed to interface RIFFA to our Karatsuba accelerator. The *packager* component manages the input stream and the *buffer* component manages the output stream. The packager temporarily stores input coefficients so they can be sent to the accelerator without interruption. This approach compensates for the risk of pipeline bursts. The buffer is responsible for improving upload transfers. It stores output coefficients from the Karatsuba algorithm until it is able to perform a complete transfer to the software without interruption. This approach addresses the issue of the input stream length being larger than the output stream length. Two more post-computations than pre-computations were implemented to reduce the number of output lanes, resulting in a reduction in the size of the output stream. For 729 input sub-polynomials with 40 coefficients (29,160 coefficients in total), our Karatsuba accelerator produces 81 sub-polynomials with 319 coefficients (25,839 coefficients in total). The bandwidth

Table 1. Hardware Resource Consumption for the FPGA-based Co-design Accelerator

		RIFFA	RIFFA/ Karatsuba interface	Karatsuba
Setup $(n, \log_2 q)$		(2560, 135)		
ALM	for logic	8,207	286	30,566
	for memory	110	0	30,030
	total	8,317	286	60,596
Registers		11,334	203	79,440
Memory bits		697,720	8,464,384	25,164
DSPs		0	0	80
f_{max}		250 MHz (PCI-E limitation)		

issue in this case is limiting because it requires a long wait before transfer initiation, leading to the use of a large FIFO.

4 IMPLEMENTATION RESULTS

Our design has been implemented on a DE5-net platform from Terasic. The platform includes a Stratix V (GXEA7N2F45C2) FPGA, several embedded memories (SRAM, Flash), and 8GB of DDR3 memory. For communication, the DE5-net provides four SPI+ connectors, a PCI-E interface (up to 8 lanes) and one RS422. Our co-design architecture also includes a desktop computer which runs the Microsoft Windows 7 operating system on an Intel Core i7-4790. The DE5-net board was plugged into one of the PCI-E slots.

Table 1 shows the FPGA hardware resources consumption for the co-design accelerator. The contribution of RIFFA, the interface between RIFFA and the Karatsuba accelerator, and the accelerator itself are noted. The Karatsuba algorithm implementation is responsible for a substantial consumption of FPGA arithmetic logic modules (ALMs). Half of these ALMs are used as memory. This is the consequence of the pipeline cost and the temporary storage of coefficients required by the architecture. Pipelining is important to reach the minimum 250 MHz frequency imposed by the PCI-E. The substantial memory requirement of the design (more than 1 MB) is due to the interface between the multiplier and RIFFA, as described in Section 3.3.6. The buffer is responsible of more than 99% of the memory consumption. This issue reveals that the main Karatsuba limitation is the length of data transfers. For the output stream, we need to send data without interruption to maximize the bandwidth, and so a large amount of data is buffered for that purpose. To reduce the memory impact, the bandwidth itself would need to be improved, possibly by reducing the load on the PCI-E bus. It is also possible to implement additional post-computations in hardware, reducing the software post-computation time. Several issues complicate the comparison of our work with the state of the art. Because a large majority of implementations target the Negative Wrapped Convolution NTT for efficiency, which is not compatible with batching for binary messages, direct comparisons are not possible. As a result, we compared our design with software algorithms from the NFLlib library. NFLlib is also based on NTT, but it is enough flexible to be adapted for batching. To speed-up computations, NFLlib splits polynomial coefficients into small numbers using the RNS system. RNS provides an efficient strategy to parallelize computations and works similarly to CRT. Table 2 compares the total computation time of this work with NFLlib and the software memory requirements. The results are given for the computation of $P_{key} \cdot U + E$ (Algorithm 1 provides more details about this computation in NFLlib). As random number and Gaussian noise generation are not in the scope of the implementation, they are not included. Our accelerator is approximatively 1.5 times

Table 2. Computation Times and Software Memory Requirements to Perform the Product/Accumulation Operation Required During the Ciphering of FV (n=2560, $\log_2 q$ = 135). Software Computations are Performed on an Intel Core i7-4790

Computation time Notation: average time (standard deviation)		
Encryptions	This work	NFLlib [20]
1	1,935 μs (220μs)	3,078 μs (98μs)
2	2,191 μs (138μs)	5,646 μs (148μs)
3	2,410 μs (176μs)	8,218 μs (191μs)
4	2,525 μs (184μs)	10,814 μs (237μs)

Software memory requirements		
	This work	NFLlib [20]
P_{key}, U, E	150 KB	480 KB
Pre-allocation	570 KB	0 KB
Total	720 KB	480 KB

ALGORITHM 1: Product/Accumulation using NFLlib

Require: P_{key} (in RNS and NTT form), U and E

1: $\widetilde{U} \leftarrow \text{NTT}(\text{RNS}(U))$
2: $\widetilde{E} \leftarrow \text{NTT}(\text{RNS}(E))$
3: $\widetilde{R} \leftarrow P_{key} \cdot \widetilde{U} + \widetilde{E}$
4: $R \leftarrow \text{invRNS}(\text{invNTT}(\widetilde{R}))$
5: **return** R

faster for 1 encryption and 4 times faster for 4 encryptions. The important drawback in NFLlib is the several computations of RNS and NTT. Karatsuba does not require such transformations.

For software memory requirements, because Karatsuba has several data dependencies between recursions, pre-allocation is required for efficiency. NFLlib also requires pre-allocated memory, but has been deported to the polynomial itself as they are mostly polynomial-dependents (NTT intermediate coefficients, RNS and NTT pre-computed constants , . . .). However, the overall cost benefits to NFLlib as very few polynomials are required during encryption.

We must also notice that a pure software implementation of Karatsuba is clearly not competitive compared to NFLlib (more than 7 ms are spent just for pre- and post-computations). Thus, the hardware accelerator significantly improves computation time. This work also greatly improves the implementation in [15], the basis for our architecture. Because the architecture in [15] does not exploit the specific structure of polynomials, the computation time of the same operation costs 2.44 ms per encryption, without any asymptotic gain for several successive encryptions.

5 CONCLUSION

In this paper, we described a high speed hardware/software accelerator to speed-up the ciphering operation of lattice-based cryptography and, in particular, the promising FV homomorphic scheme. This implementation focuses on batching techniques which pack several messages inside one ciphertext to reduce the ratio between encrypted data length and message length. We target polynomial arithmetic and compare the approach speed-up to a state of the art Lattice-Based

arithmetic software library, NFLlib. Our accelerator is approximatively 1.5 times faster for 1 encryption and 4 times faster for 4 encryptions. To achieve such performance, a high speed software library using AVX2 has been implemented, coupled with a fully pipelined hardware accelerator to reduce the transfer latency impact between software and hardware. This paper also demonstrates the interesting use of FPGAs as peripherals to improve computation times in homomorphic operations. Future work will consist of implementing the architecture on an FPGA/processor platform targeted to embedded applications.

ACKNOWLEDGMENT

This study has been partially funded by the french Direction Générale de l'Armement (DGA).

REFERENCES

[1] T. E. Gamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Proc. of CRYPTO*.
[2] P. Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proc. of EUROCRYPT*.
[3] C. Aguilar Melchor, P. Gaborit, and J. Herranz. 2010. Additively Homomorphic Encryption with D-Operand Multiplications. In *Proc. of CRYPTO*.
[4] C. Gentry. 2009. A Fully Homomorphic Encryption Scheme. Ph.D. Dissertation. Stanford University.
[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proc. of ITCS*.
[6] Z. Brakerski. 2012. Fully Homomorphic Encryption Without Modulus Switching from Classical GapSVP. In *Proc. of CRYPTO*.
[7] J. Fan and F. Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144.
[8] C. Gentry, A. Sahai, and B. Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Proc. of CRYPTO*.
[9] Z. Brakerski and V. Vaikuntanathan. 2014. Lattice-Based FHE as Secure as PKE. In *Proc. of ITCS*.
[10] A. Khedr, G. Gulak, and V. Vaikuntanathan. 2015. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Transactions on Computers* (2015).
[11] T. Lepoint and M. Naehrig. 2014. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Proc. of AFRICACRYPT*.
[12] J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. 2013. Batch Fully Homomorphic Encryption over the Integers. In *Proc. of EUROCRYPT*.
[13] J. Pollard. 1971. The Fast Fourier Transform in a Finite Field. In *Mathematics of Computation*.
[14] S. Sinha Roy, K. Järvinen, F. Vercauteren, V. Dimitrov, and I. Verbauwhede. 2015. Modular Hardware Architecture for Somewhat Homomorphic Function Evaluation. In *Proc. of CHES*.
[15] V. Migliore, M. Mendez Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat. 2016. Hardware/Software co-Design of an Accelerator for FV Homomorphic Encryption Scheme using Karatsuba Algorithm. *IEEE Transactions on Computers*.
[16] A. Karatsuba and Y. Ofman. 1962. Multiplication of Multi-Digit Numbers on Automata (in Russian). *Doklady Akad. Nauk SSSR*. Translation in Soviet Physics-Doklady.
[17] O. Regev. 2009. On Lattices, Learning With Errors, Random Linear Codes, and Cryptography. *Journal of the ACM*.
[18] P. Barrett. 1986. Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor. In *Proc. of CRYPTO*.
[19] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner. 2015. RIFFA 2.1: A Reusable Integration Framework for FPGA Accelerators. *ACM Transactions on Reconfigurable Technology and Systems*.
[20] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, and L. T. Killijian, MArc-Olivier. 2016. NFLlib: NTT-Based Fast Lattice Library.

Received April 2017; revised June 2017; accepted June 2017

5.3 Réalisation d'un outil de prototypage et d'aide à l'exploration de paramètres des schémas de chiffrement homomorphe

Les schémas de chiffrement homomorphe reposent sur des traitements multiples et coûteux. Il est alors nécessaire pour le développeur d'identifier le schéma qui correspond aux besoins applicatifs. Cette tâche n'est pas triviale. Elle nécessite de déterminer, en fonction de contraintes comme par exemple le temps d'exécution, la consommation de ressources mémoires, le niveau de sécurité ou le nombre d'opérations réalisées sur les données chiffrées, les paramètres d'un ou plusieurs schémas à des fin de comparaison. Le choix du concepteur repose alors sur son expérience et ses propres implémentations des schémas lui semblant pertinents. Cependant, le concepteur d'une application est rarement à l'aise avec les algorithmes cryptographiques avancés. C'est dans ce contexte, qu'en 2015, Cyrielle Feron a débuté ses travaux de thèse [Feron 2018] dont le but était de développer un outils d'aide au prototypage et à l'exploration de paramètres des schémas de chiffrement homomorphe.

5.3.1 L'outil PAnTHErS

PAnTHErS (*Prototyping and Analysis Tool for Homomorphic Encryption Schemes*) est un outil qui s'adresse à la fois aux experts des schémas homomorphes et aux concepteurs dont le rôle est d'intégrer un schéma au sein d'une application. Pour les experts, c'est un moyen de réaliser des analyses de schémas intégrés à l'outil, mais également d'ajouter des schémas à l'outil. Pour le non expert, PAnTHErS permet une exploration automatisée des paramètres des schémas disponibles dans l'outil afin d'extraire la configuration correspondant aux besoins applicatifs.

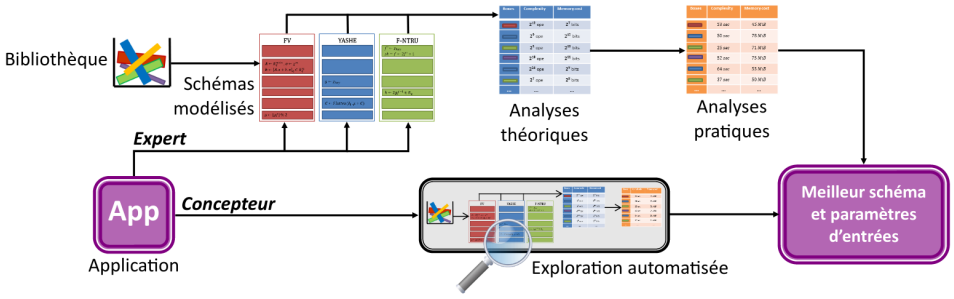


FIGURE 5 – PAnTHErS - vue d'ensemble

La figure 5 présente les différentes fonctionnalités de l'outil PAnTHErS. La première d'entre elles permet à un expert de modéliser un schéma en s'appuyant sur une bibliothèque d'algorithmes. Trois types d'algorithmes ont été proposés :

— Un **algorithme atomique** représente une opération algébrique sur un ou plusieurs objets mathématiques. Les algorithmes atomiques englobent les opérations algébriques sur les entiers, les polynômes et les matrices dont l'addition, la multiplication, la soustraction, le modulo, le tirage aléatoire et l'arrondi.

— Un **algorithme spécifique** est une combinaison d'algorithmes atomiques et/ou spécifiques. Ces algorithmes spécifiques vont être créés suite à l'étude de schémas de chiffrement homomorphe reposant sur des problèmes similaires. Des séquences d'instructions identiques entre deux schémas pourront être modélisées sous la forme d'un Spécifique. Comme une séquence d'instructions est un enchaînement d'opérations unitaires, chacune de ces opérations sera alors représentée par un algorithme atomique.

— Un **algorithme basique** correspond à l'une des cinq fonctions haut niveau d'un schéma de chiffrement homomorphe classique, c-à-d génération de clé, chiffrement, addition homomorphe, multiplication homomorphe et déchiffrement. Ces cinq fonctions peuvent être

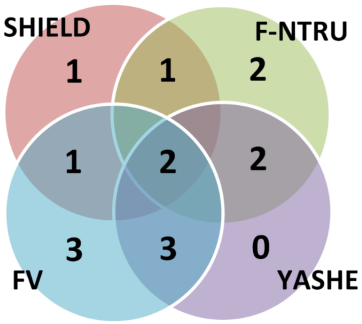


FIGURE 6 – Distribution des algorithmes spécifiques entre les schémas FV [Fan 2012], YASHE [Bos 2013], F-NTRU [Doröz 2020] et SHIELD [Khedr 2015]

complétées par d'autres algorithmes pour certains schémas, p. ex. la relinéarisation comme pour le schéma FV. Ces algorithmes serviront de base à la description d'une application. Pour cela, l'application est modélisée sous la forme d'une succession de ces algorithmes basiques.

La figure 6 illustre l'intérêt de baser la modélisation des schémas via la construction d'une bibliothèque d'algorithmes. En effet, on remarque sur cette figure que les différents schémas partagent des algorithmes spécifiques. Par exemple, l'algorithme SHIELD possède un algorithme commun au schéma FV, un algorithme commun au schéma F-NTRU et deux algorithmes communs avec les schémas FV, F-NTRU et YASHE. L'approche permet alors une modélisation accélérée de nouveaux schémas. Dans la figure 6, on remarque que le schéma YASHE peut être modélisé à l'aide d'algorithmes issus de la modélisation des schémas FV et F-NTRU.

Suite à la modélisation complète d'un schéma, une ou plusieurs analyses peuvent être menées. Le premier type d'analyse est dit *théorique*. Cette analyse permet de déterminer le nombre d'opérations effectuées par le schéma (c.-à-d. la complexité algorithmique) et/ou le nombre d'entiers stockés pendant l'exécution du schéma (c.-à-d. la consommation mémoire). Cette analyse théorique ne nécessite pas l'exécution réelle du schéma. L'utilisateur de l'outil, analyse les résultats retournés via des graphiques montrant la tendance de la complexité et/ou de la consommation mémoire des schémas au fil de l'enchainement des algorithmes utilisés. Le second type d'analyse, dite *pratique* s'appuie sur l'analyse théorique et sur quelques exécutions du schéma afin d'évaluer, pour un jeu de paramètre donné, une complexité algorithmique évaluée en secondes et une consommation mémoire exprimée en mébioctets. Pour cela, une méthode de calibration, appelé *p-calibration* a été développée durant les travaux.

La méthode de p-calibration s'effectue en plusieurs étapes. Tout d'abord, une sélection d'un nombre p de jeux de paramètres est effectuée. Ces jeux de paramètres sont sélectionnés méthodiquement afin de maximiser la couverture de l'espace de valeurs des paramètres, tout en sélectionnant, quand c'est possible, le minimum et le maximum de chaque intervalle. Une exécution concrète du schéma ou de l'application est effectuée pour chacun de ces p jeux de paramètres. Les résultats obtenus vont ainsi définir p points de référence, qui traduisent chacun le temps d'exécution et la consommation mémoire réelle du schéma ou de l'application. Enfin, le processus de calibration s'efforce de déterminer un polynôme passant par les points de référence en suivant au mieux la courbe obtenue via les analyses théoriques. À titre d'exemple, la figure 7, présente les résultats d'une calibration pour différentes valeurs de p. On observe dans ce cas, qu'une calibration avec deux points de références ($p = 2$) permet d'estimer le temps d'exécution de l'application pour une grande plage d'un paramètre donné (ici le paramètre w utilisé dans les fonctions *PowersOf* et *WordDecomp* du schéma FV).

Pour que les analyses proposées soient accessibles à un concepteur ne sachant pas comment

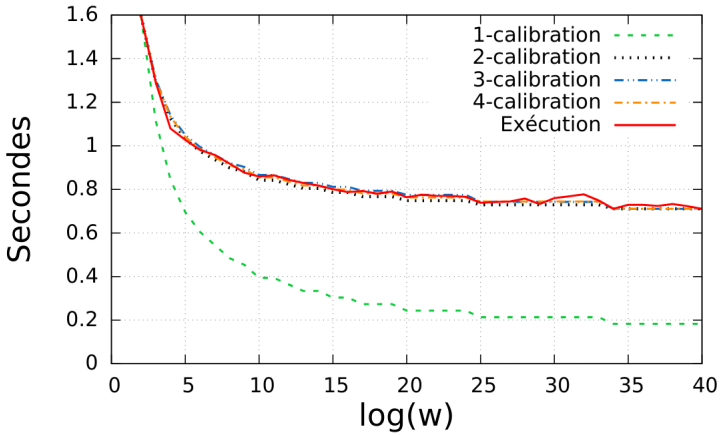


FIGURE 7 – Exemple de complexité calibrée pour pour une application enchaînant les cinq algorithmes basiques du schéma FV

TABLE 9 – Classement des points sélectionnés par l'exploration et écart avec le résultat optimal

	Classement		Écart avec l'optimal	
	Position	En %	Temps (sec)	Coût mémoire (Mio)
Pédagogique	119/22969	0,5137	0.1168	16.8
Croissant	309/5684	5,4187	1,6328	4,3
Médicale	133/5808	2,2727	16,4926	5,4

faire varier les paramètres d'entrée d'un schéma, une méthode d'exploration automatisée a été proposée. Cette méthode permet d'orienter le choix du concepteur vers un schéma et un jeu de paramètres afin qu'il soit mis en oeuvre au sein d'une application. Pour cela, le concepteur définit la profondeur multiplicative minimale (ou un interval de valeurs) et son souhait d'optimisation (temps d'exécution ou consommation de ressources en mémoire). L'outil détermine ensuite les jeux de paramètres d'entrées des schémas compatibles avec le besoin du concepteur. Puis une phase d'analyse théorique et une phase de calibration sont réalisées. Enfin, après analyse des solutions possibles, un schéma et un jeu de paramètre répondant aux critères du concepteur sont fournis. Dans le cas où le concepteur ne souhaite pas optimiser le temps d'exécution ou la consommation mémoire, alors un optimum de Pareto est considéré. La table 9 présente un exemple de résultats obtenus suite à la réalisation d'une exploration automatisée pour trois applications (détaillée dans [Feron 2020]). Cet example permet d'identifier le classement (par rapport à la configuration optimale) et l'écart en terme de performance temporelle et consommation mémoire avec la configuration optimale.

Les contributions présentées dans cette section sont détaillées dans l'article ci-dessous publié en 2020 [Feron 2020]. De plus, l'outil PAnTHErS est disponible en ligne : `https://github.com/cferon/PAnTHErS`

Automated Exploration of Homomorphic Encryption Scheme Input Parameters

Cyrielle FERON[a], Loïc LAGADEC[a], Vianney LAPOTRE[b]

[a] *UMR 6285, Lab-STICC, ENSTA Bretagne, 29806, Brest Cedex 9, France*
[b] *UMR 6285, Lab-STICC, Univ. Bretagne-Sud, F-56100, Lorient, France*

Abstract

Homomorphic Encryption (HE) aims to perform computations on encrypted data. Still in research stage, a lot of HE schemes have been created but their comparison remains costly as execution exhibits prohibitive costs. PAnTHErS is a HE schemes modeler. Modeling with a common formalism allows the evaluation of input parameters variation impact on performances of a HE scheme execution *i.e.* on its execution time and its memory cost. However, choosing a values interval that makes sense during the exploration phase still requires high expertise. This may prevent the tool from a large adoption by novice users. In this paper, we remind functionalities of PAnTHErS and present an automated exploration method. This exploration method will offer designers a simplified access to PAnTHErS functionalities.

Keywords: Homomorphic Encryption, Analysis, Modeling, Exploration

1. Introduction

Currently, Internet of things tends to increase the number of mobile data terminals as smartphones or tablets. Since their computational resources are limited, many services are deported in the cloud. This approach raises data security issues due to the fact that third parties may need to access sensitive or personal data for delivering services to the final user. Homomorphic encryption aims at answering these issues.

Homomorphic Encryption (HE) allows delegating computations on confidential data to a third party without loss of confidentiality. Figure 1 shows how a user can use a cloud computing service to modify his data with HE. User's data are secured during both transfers and computations.

To emphasis the interest of HE, let's consider a simple privacy and trust oriented example. Bob wants to use an health related application on his smartphone. He first has to enter his weight, his heart rate, etc. Then, the application (securely) transfers those data D to a server in charge of computing a personalized fitness program. In the context of classical cryptography, Bob provides his plain-data and has to trust the application. This is a major concern if he wants his data to be kept confidential.

However, HE solves this problem by keeping Bob's data D encrypted during the whole process. The considered application receives HE encrypted data $Enc(D)$ from Bob and has a function f to apply on it. This function can run on a distant server. The server processes $f(Enc(D))$ and returns the result to Bob. Bob is then able to decrypt the result by processing $Dec(f(Enc(D)))$ and then retrieves the processed data. Indeed, when a HE is used,

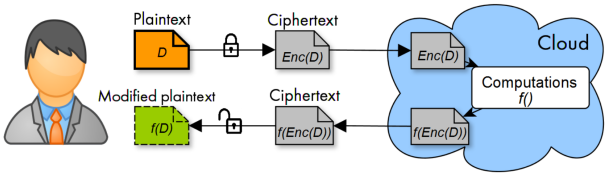


Figure 1: Homomorphic Encryption (HE) principle.

the equality

$$Dec(f(Enc(D))) = f(Dec(Enc(D))) \quad (1)$$

is true, and by definition, $Dec(Enc(D)) = D$. So, Bob recovers

$$Dec(f(Enc(D))) = f(Dec(Enc(D))) = f(D) \quad (2)$$

where $f(D)$ represents the answer of the application *i.e.* the referenced number of the proposed fitness program. On Figure 1, encryption is represented as a locked padlock and decryption by an open padlock and function f by the box *Computations*.

Unfortunately, despite all reported efforts in that field of cryptography, HE is still not usable in real cases. Indeed, expansion factor of ciphertext is important and so, length of encrypted data is huge compared to the plain data length. Moreover, significant encrypted data implies computational complexity and memory consumption issues. The cloud or the distant server must have enough resources to serve all users in terms of memory space and computational capacity. These issues prevent HE from a massive adoption despite its good properties regarding privacy, security and trust.

In this context, the adoption of HE relies on the capacity of the application designer to identify the scheme that

best fits a given application. For that purpose, several criteria are fixed by the application and server requirements. These criteria can include maximal execution time, multiplicative depth *i.e.* the maximal number of homomorphic operations that can be processed on encrypted data, memory space and security.

Due to memory and complexity issues, the analysis of all HE schemes in the literature implies considerable software executions time. For instance, in FV scheme [1], key generation and encryption of one bit takes 0.044 seconds with the tool Cingulata [2]. Encrypted result (resp. keys) has a size of 6 153 bytes (resp. 20.4 KB). On the other side, AES-CBC encryption with OpenSSL 1.0.1t takes less than 0.001 seconds to return a ciphertext (resp. key) of 32 bytes (resp. 16 bytes). Furthermore, design space exploration of homomorphic schemes means varying the input parameters of the HE scheme, which are at least three. As HE is very memory and time consuming, HE analysis by software simulations is not scalable. Besides, some HE schemes have several variants making exploration even more time consuming.

In this work, the tool PAnTHErS is presented: PAnTHErS helps the designer analyzing such HE schemes. Already usable by HE experts, PAnTHErS is extended by an exploration method for non-specialists. This exploration method allows to find the most interesting scheme and input parameters optimizing both complexity and memory cost for a given application.

The remainder of this paper is organized as follows. Section 2 presents some related work. Section 3 introduces the tool PAnTHErS. Then, Section 4 details the exploration method we implemented in PAnTHErS. Section 5 applies exploration method on four HE applications including one in a medical context. Section 6 recaps the different usages of PAnTHErS. Finally, Section 7 draws some conclusions and presents future work.

2. Related Work

In 1978, Rivest et al. [3] introduced the problem of creating a homomorphic encryption scheme which permits computations on encrypted data. At the time, they talked about *privacy homomorphism*. On the one hand, RSA cryptosystem [4] created by Rivest, Adleman and Shamir appears to be multiplicatively homomorphic. On the other hand, some schemes are additively homomorphic like Paillier cryptosystem [5]. Interest of full HE is to be multiplicatively and additively homomorphic. In 2009, Gentry constructed the first Fully Homomorphic Encryption (FHE) [6], which is based on ideal lattices. Since Gentry's work, numerous HE schemes have been created. FHE is achieved with Somewhat Homomorphic Encryption (SHE) scheme (bounded computation depth) combined with bootstrapping technique.

A SHE scheme only has a limited number of multiplications that can be applied on encrypted data: this number is called the multiplicative depth. Essentially, HE schemes rely on summing a noise to plaintext to produce a ciphertext. The final result can be decrypted only if the noise is below scheme specific threshold. Thus, noise expansion has to be controlled. In 2009, Gentry [6] used, on a SHE scheme, the bootstrapping technique that decreases (or "refreshes") noise in ciphertexts, without requiring the plaintext. He obtained a FHE scheme which can produce an unlimited number of multiplications.

More efficient noise management techniques have been introduced for HE schemes as modulus switching [7]. Also, ciphertext maintenance is needed during computations in order to preserve consistency (key switching [7], scale invariance technique [8], dimension modulus reduction [9], relinearization [9]). Moreover, SHE was combined with symmetric-key encryption to make trans-ciphering [10] in order to reduce the size of encrypted data that has to be transferred to the remote server.

Not all created schemes rely on ideal lattices like the one by Gentry. They are based on different hardness assumptions. HE schemes over the integers [11, 12, 13, 14, 15] are all based on the *approximate-GCD* problem. The idea of the *approximate-GCD* problem is to find the approximate common divisor of n random integers which are close multiples of the same large integer.

HE schemes on lattices are mainly based on *Learning With Error (LWE)* [16, 9, 8] and *Ring-LWE* (R-LWE) [17, 7, 1] problems. Introduced by Regev [18] in 2005, LWE problem is : given q and n, find $s \in \mathbb{Z}_q^n$ from "approximate" random linear equations on s. It means that each equation of s is perturbed by a small amount of noise. The R-LWE problem, also introduced by Regev [18], is basically to shift from $\mathbb{Z}_q^n$ to the ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ in LWE problem definition. These two problems are post-quantum -meaning they resist to an attack by a quantum computer- what makes them very interesting in cryptography.

A method for LWE-based HE schemes, called approximate eigenvector, was introduced by Gentry, Sahai and Waters in 2013 [19]. In their scheme, the secret key is an approximate eigenvector of the ciphertext and the message is the eigenvalue. Error is introduced in eigenvector to enforce the security of the scheme on LWE. [20] is also based on this method. Introduced by Hoffstein, Pipher and Silverman in 1998 [21], NTRU is one of the first public-key cryptosystems based on lattices. It was then taken in reference to make HE schemes by [22, 23].

Some implementations of HE are now open-sources. HElib [24] is a well-known implementation of Brakerski, Gentry and Vainkuntanathan's scheme. HElib is a software C++ library, implemented by Halevi and Shoup. Also, a implementation of [11] by Coron et al. is available in Sage [25]. Aguilar-Melchor et al.[26] implemented a C++ library called NFLlib [27]. This is a library for ideal lattices cryptography in the polynomial ring $\mathbb{Z}_p[x]/\langle x^n+1\rangle$. This library allows the creation of FV-NFLlib [28] which is a implementation of [1] using NFLlib.

This section demonstrates that the quest for a practical

HE scheme has implied the creation of numerous schemes having relatively similar properties. Still, computational complexity and memory cost are the main issues of HE. Despite HE schemes similarities, no generic method allows analyzing or comparing HE schemes. Indeed, analyzing HE schemes means executing them, leading to a prohibitive cost while performing exploration and optimization tasks. To deal with this issue, this paper proposes to extend the PAnTHErS tool with an automated exploration of HE scheme input parameters allowing the extraction of an optimized configuration for a given application.

3. PAnTHErS

One limitation of HE is the hardness to estimate HE schemes complexity and memory cost and so, comparing them. Analysis requires HE experts to implement the scheme (whichever the used language and the fixed target architecture). Once implemented, which implies costly design and debug phases, numerous executions are still required to explore several input parameters impact.

PAnTHErS is an all-in-one solution to cut these costs. First, PAnTHErS offers a library to easily integrate HE schemes, alleviating the need for starting implementation from scratch. This strongly promotes legacy reuse. Also, thanks to a calibration phase, PAnTHErS provides facilities for rapid evaluation which only require a limited number of executions. Those evaluations allow the user to detect costly operations, highlighting algorithm sections on which refactoring effort must focus. Also, analysis makes it easier to choose which input parameters to concentrate on – if not proper values to select - according to implementation design and user's application needs. In addition, PAnTHErS allows comparison of schemes, and provides visual feedback, producing graphs showing complexity and memory cost. Thus, not only does the tool help HE experts to analyze and compare schemes but, it also meets the needs of designers, new to cryptography, wishing to use HE in future applications.

Before analyzing HE schemes, schemes must first be modeled into a common format. PAnTHErS (Prototyping and Analysis Tool for Homomorphic Encryption Schemes) aims at answering to this problem. The tool, illustrated in Figure 2, is divided in several steps including a modeling step, a theoretical analysis step and a calibration step. This section presents those different steps which are fully detailed in [29, 30].

3.1. Modeling

HE schemes are first modeled into a succession of algorithms. Three types of algorithms exist: Atomics (each one represents one operation), Specifics (representing series of instructions *i.e.* series of Atomics and/or Specifics) and HE basics. HE basics represents the five functions of a HE scheme: key generation, encryption, homomorphic addition, homomorphic multiplication and decryption. Specifics are composed of instructions which are identical in several HE schemes and thus, they are usable in different modeled schemes.

The created algorithms are stored in PAnTHErS library to be used in several scheme modeling. Then, the more the library possesses algorithms, the more it is possible to reuse algorithmic sequences, allowing to speed up modeling of new schemes.

Thanks to the modeling method (detailed in [29]), modeled schemes can be analyzed on a common basis. From models, PAnTHErS evaluates both computational complexity and memory cost of HE schemes. Analysis results allow HE scheme comparison in order to find the best scheme (*i.e.* implying the less complexity and/or the lowest memory consumption).

3.2. Theoretical analysis

Scheme theoretical analysis, deeply described in [29], is made in the worth case. Each algorithm A of PAnTHErS library *i.e.* Atomic, Specific and HE basic, is linked to two evaluation modules: computational complexity and memory consumption.

Complexity analysis module follows complexity evolution by counting the number of operations that are performed in A. In the case of memory analysis module, memory cost evolution is followed by listing (and updating the list) containing information about variables that are created or modified by A. At the end of an analysis, complexity and memory analysis modules return respectively the maximal number of multiplications performed during scheme execution and the maximal number of 32-bit integers simultaneously stored during scheme execution.

With these modules, a HE scheme can be analyzed without requiring its concrete execution. Results which are returned by theoretical analysis show evolution of computational complexity and memory cost in function of HE scheme input parameter variations. These results provide input parameter influence on the analyzed HE scheme flow. Moreover, it is also possible to observe computational complexity and memory cost after each call of algorithms that compose the scheme. It highlights costly operations (for instance, multiplication of polynomials), storage of temporary variables and outputs in a scheme (for example, keys, plaintexts and ciphertexts). Thus, this information could help designer while optimizing their scheme implementation.

However, theoretical analyses highlight input parameters influence on a scheme without reflecting a concrete scheme execution on a given implementation. Consequently, it is hard to estimate the scheme execution time from theoretical analysis of complexity. Moreover, theoretical analyses which are made in the worst case do not provide a satisfying representation of parameter concrete influence during practical usage of HE schemes. For instance, worst case analysis can show that a scheme S_1 has a less important complexity than a scheme S_2. Nonetheless, it is possible that S_1 execution will be longer than S_2 execution. This situation is not sufficient because PAnTHErS

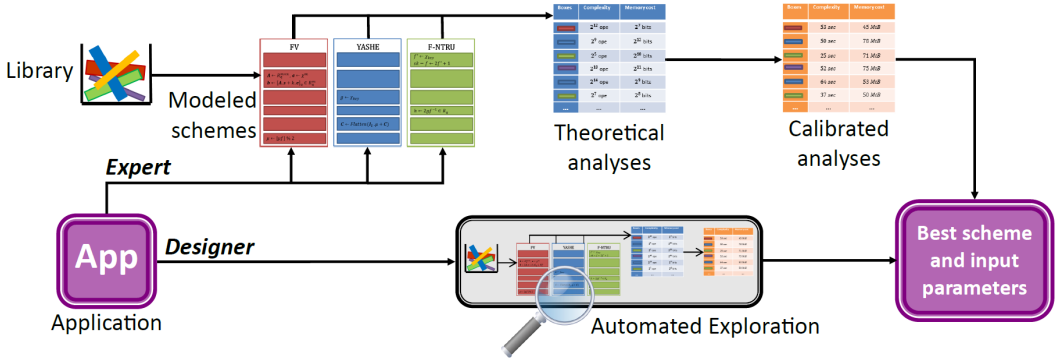


Figure 2: PAnTHErS overview.

aims at comparing several schemes between them. To do so, we provided a module to estimate execution time of schemes from theoretical analyses: this module is named p-calibration.

3.3. Calibration

p-calibration [30] allows converting computational complexity in execution time and performing evaluation of memory consumption in mebibytes (MiB). p-calibration is based on concrete executions of p sets of parameters called referencing points. A p-calibration needs p concrete scheme executions with the chosen referencing points. Then, computational complexity (resp. memory consumption) first expressed in number of multiplications (resp. number of 32-bits integers stored) is converted into execution time in seconds (resp. in MiB), based on concrete execution results.

2-calibration definition for complexity: Let p_1 and p_2 be referencing points such as $p_i = (CompTheo_i, CompPrac_i)$ with $CompTheo$ theoretical complexity analysis and $CompPrac$ execution time obtained after scheme execution. Referencing points are chosen considering the analyzed chosen values interval I of one scheme input parameter. Referencing point p_1 (resp. p_2) then corresponds to the complexity analysis of the parameter having the lowest value (resp. highest value) in I.

2-calibration consists in finding y_1 and y_2 variables such as:

$$\begin{cases} CompTheo_1 \times y_1 + y_2 = CompPrac_1 \\ CompTheo_2 \times y_1 + y_2 = CompPrac_2 \end{cases} \quad (3)$$

Then, all theoretical analyses $CompTheo_i$ linked to each value in I (including $CompTheo_1$ and $CompTheo_2$) are calibrated by computing $y_1 \times CompTheo_i + y_2$.

Error rate can be calculated between p-calibrated results and concrete execution times (resp. MiB). By taking a bigger p, error rate can be reduced but p-calibration execution is increased. Calibration experiments [30] showed that for $p = 2$, 3 and 4:

- Average of error rates does not exceed 10 % for $p = 2$, 7 % for $p = 3$ and 8 % for $p = 4$.
- At least 95 % of calibrated results have an error rate inferior to 20 % for $p = 2$, 12 % for $p = 3$ and 14 % for $p = 4$.

Considering those error rates and p-calibration execution time, $p = 2$ is used in the exploration step (section 4). A user who prefers lower error rates (despite a greater p-calibration execution time) can take a larger p.

When analyzing input parameters, the application designer has to carefully select set of parameters that respect the multiplicative depth required by the targeted application. This task requires a good knowledge of HE schemes. Thus, the designer has to be supported by a HE expert.

In order to make PAnTHErS usable by both HE expert and designers, we extend the tool with an exploration phase presented in the next section.

4. Exploration

The analysis of HE schemes is realized by the tool PAnTHErS [29]. First, a modeling step is necessary to represent a scheme into a succession of functions. Authors of [29] explained how a HE expert can process theoretical analyses on computational complexity and memory cost, while in [30], they show calibration procedure which calibrates theoretical analyses into concrete results that are expressed in seconds for execution time and mebibytes for memory consumption.

This section introduces an input parameter exploration method of HE schemes for a given application. This exploration method allows experts and non-experts in cryptography to elect the scheme along with input parameters that better fit the given application *i.e.* resulting the lowest execution time and/or the weakest memory consumption.

4.1. Exploration description

Exploration method consists in using modeling, analysis and calibration methods, as explained in the previous sections, in order to either reach the lowest cost in execution time and memory usage or to trade between the

two metrics. Exploration is divided in five steps as follows.

Step 1: *Application selection*

This step fixes an application requirement for the exploration. Concretely, the user defines the minimum multiplicative depth d required by the considered application. Alternatively, an interval of values for the multiplicative depth can be defined. During the exploration, configurations that do not provide the minimum required depth are automatically discarded.

The following steps (2, 3 and 4) are automatically executed for every modeled scheme available in PAnTHErS library. In our case, available schemes are FV [1], YASHE [22] and F-NTRU [23][1].

Step 2: *Find sets of parameters*

From its input parameters, a scheme implies a fixed multiplicative depth. Depth calculation is different for each HE scheme. Indeed, depth depends on generated noise in schemes. Authors give, in their articles, maximal bounds produced after: an encryption, a homomorphic addition and a homomorphic multiplication. These bounds allow calculating depth of the scheme from a fixed set of input parameters. Depth calculations of FV, YASHE and F-NTRU have been integrated in exploration method of the PAnTHErS tool.

Step 2 consists in finding each set of input parameters of the scheme S that implies the depth d of the application (or one depth in the interval of values chosen in step 1). For that, multiplicative depth of each set of input parameters is calculated. The sets of parameters which imply a depth from the ones chosen in step 1 are stored in a list named L.

Step 3: *Theoretical analyses*

List L, created in step 2, contains sets of valid parameters. Each set of parameters of L is theoretically analyzed as presented in 3.2.

Step 4: *Calibration*

Theoretical analyses are then calibrated with p-calibration method. p-calibration starts by executing p times the application with scheme S considering different sets of input parameters. Then, theoretical analyses are converted in concrete results (computational complexity is converted in execution time in seconds and memory consumption in mebibytes).

During step 4, only 2-calibrations are executed. Indeed, $p = 2$ has been chosen due to its acceptable produced error rates [30]. Moreover, a low value of p limits number of required application executions and thus it allows performing a faster exploration by reducing execution time of step 4.

Step 4.1: *Execution of the application with p referenced points*

In this step, the way we gather actual execution results depends on the number of varying input parameters.

General case with n variable parameters: Let $\mathcal{J}_2$ be the set which gathers sets of parameters named $j_i = (p_1, p_2, ..., p_n)$ which are chosen in step 2. Let $\forall k \in \{1, ..., n\}$, $min(p_k) = min\{p_k \in \mathcal{J}_2\}$ and $max(p_k) = max\{p_k \in \mathcal{J}_2\}$ be the minimum and the maximum of each parameter used in $j_i \in \mathcal{J}_2$. The executed sets of parameters are the sets corresponding to one combination of $min(p_k)$ and/or $max(p_k)$. Consequently, application will be executed with 2^n different sets of parameters j_{exec}. The sets of parameters j_{exec} are not necessary included in $\mathcal{J}_2$. Afterwards, $\mathcal{J}_{exec}$ represents the set of all j_{exec}.

F-NTRU case with $n = 2$: Parameters q (ciphertexts module) and w (integer for words decomposition) are input parameters in F-NTRU scheme. All $j_i \in \mathcal{J}_2$ have $(p_1, p_2) = (q, w)$ form. Minimum and maximum of each parameter are found: $min(q), max(q), min(w)$ and $max(w)$. Application is then executed with $2^n = 2^2 = 4$ different sets of parameters which are listed in equation 4. The sets of parameters j_{exec} in equation 4 are not necessarily included in $\mathcal{J}_2$.

$$\begin{aligned} A &= j_{exec_1} = (min(q), min(w)) \\ B &= j_{exec_2} = (max(q), min(w)) \\ C &= j_{exec_3} = (max(q), max(w)) \\ D &= j_{exec_4} = (min(q), max(w)) \end{aligned} \tag{4}$$

FV and YASHE case with $n = 3$: In addition to q and w, parameter t (plaintexts module) is a variable in FV and YASHE schemes. Each $j_i \in \mathcal{J}_2$ has $(p_1, p_2, p_3) = (q, w, t)$ form. Application is then executed with $2^n = 2^3 = 8$ different sets of parameters which are listed in equation 5.

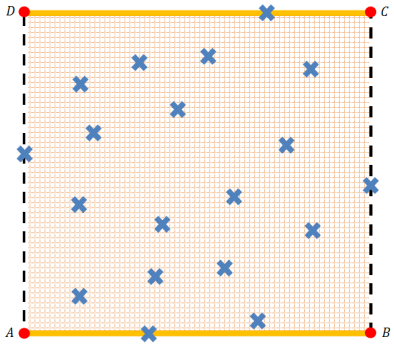
$$\begin{aligned} A &= j_{exec_1} = (min(q), min(w), min(t)) \\ B &= j_{exec_2} = (min(q), max(w), min(t)) \\ C &= j_{exec_3} = (max(q), max(w), min(t)) \\ D &= j_{exec_4} = (max(q), min(w), min(t)) \\ E &= j_{exec_5} = (min(q), min(w), max(t)) \\ F &= j_{exec_6} = (min(q), max(w), max(t)) \\ G &= j_{exec_7} = (max(q), max(w), max(t)) \\ H &= j_{exec_8} = (max(q), min(w), max(t)) \end{aligned} \tag{5}$$

Step 4.2: *Conversion of theoretical analyses*

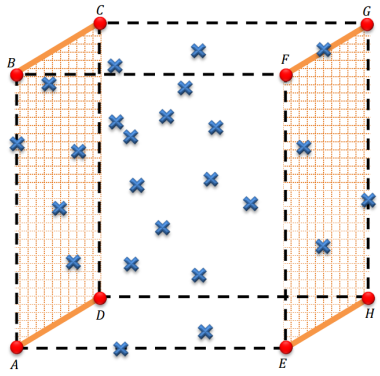
After executing sets of parameters in $\mathcal{J}_{exec}$, theoretical analyses of sets of parameters in $\mathcal{J}_2$ are then calibrated. Accordingly, the applied process for calibration is represented in Figure 3.

Figure 3a (resp. Figure 3b) corresponds to $n = 2$ case (resp. $n = 3$ case). In the two figures, blue crosses refer to sets of parameters in $\mathcal{J}_2$ and red points to sets of parameters in $\mathcal{J}_{exec}$. Coordinates of each red point are written in Figures 3a and 3b. Dotted and solid lines symbolize

[1] YASHE and F-NTRU are not secure since [31]. However, it does not impact on this paper contributions, especially on exploration feasibility.



(a) Representation for 2 parameters variation (F-NTRU).



(b) Representation for 3 parameters variation (FV and YASHE).

Figure 3: Representation of sets of parameters in $\mathcal{J}_2$, represented by blue points, and their calibration. Red points represents sets of parameters in $\mathcal{J}_{exec}$.

limits of the set $\mathcal{J}_2$. The goal, here, is to calibrate blue crosses within limits, thanks to their theoretical analysis and concrete application executions of each red point (*i.e.* each set of parameters drawn as red point).

$n = 2$ **case:** On Figure 3a *i.e.* in the case where $(p_1, p_2) = (q, w)$, application is executed for each of the four sets of parameters represented by red points. Two first calibrations are processed between :

• point $A = (min(q), min(w))$ and point $B = (max(q), min(w))$,

• point $D = (min(q), max(w))$ and point $C = (max(q), max(w))$.

These two first calibrations are presented in Figure 3a by an solid orange line (two opposite sides of the square). During the two calibrations, w is fixed and q is varied. In order to realize these calibrations, theoretical analyses of all sets of parameters included between A and B and between D and C must be first calculated.

Finally, from the four executions and the calibrated analysis, every theoretical analyses of blue crosses, *i.e.* sets of parameters in $\mathcal{J}_2$, can be calibrated. Each theoretical analysis of a set of parameters (q_i, w_i) is calibrated by taking $(q_i, min(w))$ and $(q_i, max(w))$ as referencing points. These last points correspond either to concrete results that are found after an execution or correspond to calibrated analyses. On Figure 3a, when analyses of red points and crosses on solid lines are calibrated, then theoretical analyses of all blue crosses on squared area can be calibrated. In fact, two 2-calibrations are processed successively: blue crosses calibration is executed after the first calibrations between A and B and between D and C.

$n = 3$ **case:** On Figure 3b *i.e.* in the case where $(p_1, p_2, p_3) = (q, w, t)$, application is executed with scheme by taking each of the eight red points. By fixing $p_3 = min(t)$ (resp. $p_3 = max(t)$) and by varying only q and w, theoretical analyses of points in $ABCD$ face (resp. opposite face $EFGH$) represented in Figure 3b can be calibrated as explained in $n = 2$ case. Before the calibration, theoretical analyses of all points in the two faces must be calculated. At this point, every theoretical analyses of points of (q_i, w_i, t_i) form with $t_i = min(t)$ or $t_i = max(t)$ are calibrated. Finally, each theoretical analysis of sets of parameters of (q_i, w_i, t_i) form with $t_i \neq min(t)$ and $t_i \neq max(t)$ can be calibrated by taking $(q_i, w_i, min(t))$ and $(q_i, w_i, max(t))$ as referencing points. In this case, three 2-calibrations are processed successively: blue crosses calibration is executed after $ABCD$ and $EFGH$ face calibration (corresponding to a $n = 2$ case calibration).

Step 5: *Sorting and electing of best scheme and input parameters*

Previous steps allow to get calibrated analysis of application for all schemes and for each set of parameters implying one of the multiplicative depths chosen in step 1 (either depth of the application, either one value from the chosen interval). The aim of step 5 is to elect the scheme and its sets of parameters implying best results in terms of execution time and/or memory consumption for the chosen application.

Beforehand, user can choose between optimizing lowest execution time or memory cost. In that case, parameters are sorted in function of their associated calibrated complexities (resp. memory cost). Then, the set of parameters which has the lowest execution time (resp. memory cost) is returned to the user.

If the user has no preference between complexity and memory cost, then Pareto efficiency [32] is applied as described in Definition 1.

Definition 1 (Pareto frontier in exploration method context)**.** *Let $\mathcal{J}_2$ be the set gathering all explored sets of parameters and $\mathcal{J}_P$ the set of points on Pareto frontier. Each $j \in \mathcal{J}_2$ (resp. $j_P \in \mathcal{J}_P$) produces an execution time t_j (resp. t_P) and a memory cost m_j (resp. m_P). Then $\forall j \in \mathcal{J}_2$ such as $j \notin \mathcal{J}_P$, it exists $j_P \in \mathcal{J}_P$ such as $t_P \leq t_j$ and $m_P \leq m_j$.*

Figure 4 gives an example of the Pareto frontier. Each set of parameters, represented by a square or a triangle, is placed depending on application execution time and application memory cost it implies. From Definition 1, sets of parameters on Pareto frontier are those illustrated by a

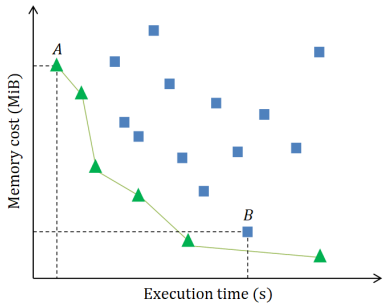


Figure 4: Graphical representation of a Pareto efficiency example. Each triangle and square represents a set of parameters in function of its execution time and memory consumption. Green triangles are sets of parameters on Pareto frontier.

green triangle on Figure 4. Graphically, it is possible to verify that a set of parameters is on Pareto frontier or not. For instance, the set of parameters A is on Pareto frontier because no other set of parameters are on the area limited by $(0,0)$, $(t_A,0)$, (t_A,m_A) and $(0,m_A)$. This means that no other set of parameters implies an execution time and a memory consumption lower than A. On the other hand, on Figure 4, a blue square is on the area limited by $(0,0)$, $(t_B,0)$, (t_B,m_B) and $(0,m_B)$; thus, set of parameters B is not on the Pareto frontier.

The sorting of the step 5 consists in finding sets of parameters that are on the Pareto frontier. These sets of parameters are then suggested to the user who will choose one of them by favoring the best execution time or the lowest memory cost.

4.2. Conclusion

The five steps of our exploration method do not require user intervention. User only gives application to be analyzed (step 1). Then, exploration analyzes all combinations of schemes and sets of parameters which respect the application requirement (*i.e.* multiplicative depth) in order to find the ones which imply the most interesting execution times and memory costs. Pareto efficiency is used to elect the best sets of parameters among those analyzed. These sets of parameters and their estimations are then shown to the user.

5. Use cases

Previous section describes the automated exploration method. Exploration allows a non-expert in HE to identify the most interesting scheme (implying the lowest execution time and/or memory cost) for a given application.

Exploration can be applied on any kind of application. In order to show exploration capabilities, four applications have been considered, each one with a different numbers of operations and a different multiplicative depth.

The first part of this section introduces the four applications while the second part presents the results obtained by applying our proposed exploration method.

5.1. Applications definition

Four applications have been created in order to process exploration method on different use cases. Applications exhibit different numbers of operations and different multiplicative depth. Applications features are detailed in Table 1.

Table 1: Applications information (number of called HE basic and depth)

Applications	*FiveHB*	*Pedagogic*	*Croissant*	*Medical*
# of *KeyGen*	1	1	1	1
# of *Encrypt*	1	3	35	235
# of *Add*	1	5	80	294
# of *Mult*	1	12	14	106
# of *Decrypt*	1	1	8	4
Total number of HE basics	5	22	138	640
Depth	1	10	6	12

5.1.1. Artificial applications

FiveHB application is a succession of the five HE basic functions in the following order: key generation *KeyGen*, encryption *Encrypt*, homomorphic addition *Add*, homomorphic multiplication *Mult* and decryption *Decrypt*.

Pedagogic application has an arbitrary number of homomorphic operations. It does not process any useful functionality for a user. It was only created to evaluate an application having a more important depth than *FiveHB* application. *Pedagogic* application thus has a depth of 10 against 1 only for *FiveHB*.

Pedagogic application has, as input parameters, three plaintexts which are polynomials. The three plaintexts m_1, m_2 and m_3 are encrypted as c_1, c_2 and c_3. Then, the following operations are processed:

$$c = c_1^3 \cdot c_2^2 \cdot c_3^2 \cdot S \cdot T \cdot (S + c_3) \cdot (S + c_2 + c_3) \qquad (6)$$

with $S = c_1 \cdot T + c_1 + c_2$ and $T = c_1 \cdot c_2 + c_3$. Result c is then decrypted.

5.1.2. Concrete applications

Croissant application allows to know how many decades of kcal a person will consume by eating n croissants. Parameter n is considered has an 8-bit input parameter.

The computations performed by this application are simple: knowing that eating a croissant implies consuming 231 kcal, only a multiplication is necessary to obtain the final quantity. Moreover, to make easier computations, application rounds quantities to the decade of kcal. Thus, a croissant is equivalent to 23 decades of kcal. Final application consists in taking as input a quantity n of croissants and then, this value is multiplied by 23.

FV [1] and YASHE [22] schemes have a variable named t which is the plaintexts module (each plaintext is reduced modulo t). This parameter is one explored parameter in exploration method. Final result of *Croissant* application must be an integer greater than 23. Thus, if t is chosen

Algorithm 1 C++ source code of *Croissant* application

```
1: Integer8 nb_croissant, c;
2: cin>>nb_croissant;
3: c = nb_croissant*Integer8(23);
4: cout<<c;
```

small (*e.g.* $t = 2$), application will not decrypt the good value because final result is reduced modulo t. On an other hand, if t is chosen large (> 23), it must be taken larger enough to correctly respond to the user. For instance, if the user eats 5 croissants, application must return the value $5 \times 23 = 115$ and so, scheme must use $t > 115$. Consequently, in order to guarantee a good functioning, the chosen value of t limits the maximal value of parameter n which is input by the user.

A method to overcome this limitation is to perform homomorphic operations on an application input bits. By directly working on bits, there is no restriction on input parameters. Nevertheless, this implies that t parameter must always be equal to 2 because every output message will be a single bit (either 0, either 1).

In order to apply this method, an application has to be converted into a boolean circuit. The tool Cingulata [2] allows to convert a C++ implemented application into a boolean circuit. This circuit is only composed of OR, XOR, AND and NOT gates. The application circuit is then saved into a BLIF (Berkeley Logic Interchange Format) file [33].

Others tools exist and can be use to perform a conversion from high level application (C/C++ language) to boolean circuit including, for example, the tool Madeo [34].

We implemented *Croissant* application in C++ as described in Algorithm 1.

The tool Cingulata gives boolean circuit of *Croissant* application from its C++ description. Cingulata returns an optimized and a non-optimized version of the boolean circuit. We consider the optimized version that we convert into a succession of HE basics, that is suitable for PAnTHErS. For that purpose, BLIF format is converted in succession of XOR, AND, NOT and OR gates. Then, each XOR gate (resp. AND) is converted in *Add* function (resp. *Mult* function). Furthermore, NOT(a) = a XOR 1 is converted in *Add*(a, *Encrypt*(1)) and OR(a, b) = AND(AND(a,b), XOR(a,b)) is converted in *Mult*(*Mult*(a,b), *Add*(a,b)).

Number of HE basics highly increases when *Croissant* application is converted in a boolean circuit. Indeed, C++ *Croissant* application calculates one multiplication between two 8-bit integers. In homomorphic domain, this application has a succession of 138 HE basics. If we take 16-bit integers as inputs, the application would perform 262 HE basics, roughly twice HE basics with 8-bit integers.

However, 8-bit integers usage reduces number of possible values for parameter n (number of croissants). As it is implemented on 8 bits, *Croissant* application result will not exceed 256 decades of kcal, that is to say 11.13 croissants. Thus, in this use case, n is limited to 11.

Algorithm 2 *Medical* application pseudo-code (described in [35] and available in [2])

Require: gender, age, medicalHistory, smoker, diabetic, HDLcholesterol, highBloodPressure, weight, alcoholConsumption, physicalActivity, height
Ensure: riskFactor (between 0 and 9)

```
1: riskFactor = 0
2: If gender == MAN AND age > 50:
3:         riskFactor += 1
4: If gender == WOMAN AND age > 60:
5:         riskFactor += 1
6: If medicalHistory:
7:         riskFactor += 1
8: If smoker:
9:         riskFactor += 1
10: If diabetic:
11:         riskFactor += 1
12: If highBloodPressure:
13:         riskFactor += 1
14: If HDLcholesterol < 40:
15:         riskFactor += 1
16: If daily physicalActivity < 30 minutes:
17:         riskFactor += 1
18: If weight - 10 ¿ height:
19:         riskFactor += 1
20: If gender == MAN AND alcoholConsumption > 3
    glasses/day:
21:         riskFactor += 1
22: If gender == WOMAN AND alcoholConsumption >
    2 glasses/day:
23:         riskFactor += 1
```

Medical application is presented in [35]. This application aims at computing a risk factor of cardiology disease from medical information of a patient. Calculated risk factor is an integer between 0 (weak risk) and 9 (strong risk). *Medical* application pseudo-code is written in Algorithm 2.

As for *Croissant* application, *Medical* application has been converted into a boolean circuit by the tool *Cingulata* [2] and t is fixed to 2 for equivalent reasons.

5.2. Application exploration results

This section introduces exploration method results of three schemes (FV [1], YASHE [22] and F-NTRU [23]) on the four applications: *FiveHB*, *Pedagogic*, *Croissant* and *Medical*. Hardware and software configuration for experiments is showed in Table 2. For each exploration, we have allocated 2 cores and 128 GB of RAM. The Sage 8.0 [36] open-source computer program has been launched in a Docker container. The ranges of parameters exploration for each scheme of each application are defined in Table 3.

Table 2: Hardware and software configuration of exploration execution environment.

Processors	8 × Intel Xeon E7-8890 v4 @ 2.20 GHz
Number of cores	8 × 24
RAM	192 × 32 Gb / DDR4
Operating system	Red Hat Enterprise Linux 7.5 64-bit
Software environment	Sage 8.0 Python 2.7.10

Table 3: Minimum and maximum bounds reached by input parameters of each scheme.

Application	**Scheme**	$\log(q)$		$\log(w)$		t	
		Min	**Max**	**Min**	**Max**	**Min**	**Max**
FiveHB	FV	43	148	2	99	2	49
	YASHE	45	162	2	99	2	49
	F-NTRU	18	124	2	99	-	-
Pedagogic	FV	297	418	2	99	2	14
	YASHE	308	443	2	99	2	5
	F-NTRU	102	149	2	7	-	-
Croissant	FV	189	287	2	99	2	2
	YASHE	196	308	2	99	2	2
	F-NTRU	61	149	2	19	-	-
Medical	FV	376	476	2	99	2	2
	YASHE	389	499	2	99	2	2
	F-NTRU	121	149	2	4	-	-

Table 4 gives number of explored parameters, exploration execution time and required time to perform all concrete executions for each application. A speedup factor is defined for each application. This factor is the ratio between concrete execution time and our exploration execution time.

Speedup factor ranges from 7.4 (*FiveHB*) to 41.8 (*Pedagogic*). Speedup of all gathered applications is 18.6. A study per scheme of application speedup factor shows, in Table 5, that in general F-NTRU (resp. YASHE) has the lowest (resp. the highest) yield. For instance, speedup factor of F-NTRU for *Medical* application is 5.9 and the one of YASHE for the same application is 137.5. This difference is mainly due to the number of analyzed sets of parameters (only 26 for F-NTRU against 2936 for YASHE) and to concrete execution times that are required for calibration: in average, an execution of *Medical* application with F-NTRU scheme is 53 times slower than with YASHE scheme.

5.2.1. Error rates study

Figure 5 illustrates spreading of error rate that were noted between calibrated analysis of exploration and concrete executions. Error rates of Figure 5 are showed by application exploration *i.e.* all schemes combined.

More than 95 % of error rates, for each application, are under 45 %. Moreover, the average of error rates for calibrated complexity (resp. memory cost) is :

- 14.7 % (resp. 16.6 %) for *FiveHB* application,
- 7.5 % (resp. 3.8 %) for *Pedagogic* application,
- 7.4 % (resp. 5.1 %) for *Croissant* application,
- 7.5 % (resp. 1.3 %) for *Medical* application.

All those means combined are under 17 %. Considering application error rates for each separated scheme, the observations on error rates above are also verified except for *Croissant* application with F-NTRU scheme. For this specific case, more than 50 % of error rates produced by the exploration are superior to 45 % for calibrated complexity. Regarding memory cost, only 70 % of values are actually inferior to 45 %.

Those error rates can be decreased by executing application with more referencing points. As we took $p = 2$, the number of referencing points is limited to 4 for F-NTRU and 8 for FV and YASHE (see section 4) but it could be higher, depending on the total number of sets of parameters to explore or depending on the size of interval of varied parameters. By having a large number of referencing points, p-calibration will be thinner and thus will induce lower error rates. However, the more there is referencing points, the longer will be the exploration execution.

5.2.2. Optimal sets of parameters study

Sets of parameters obtained by application exploration are listed in Table 6. Selected sets of parameters by exploration are for YASHE scheme in the case of *FiveHB* application and for FV scheme in the case all the three other applications.

Optimal sets of parameters are obtained after concrete execution of all schemes for each application. As for selected sets of parameters, those optimal points are sets of parameters for YASHE in the case of *FiveHB* application and for FV scheme in the case all the three other applications. FV scheme offers a faster calculation and less memory consumption than YASHE and F-NTRU scheme for more realistic applications.

Figure 6 illustrates spreading of analysis of explored sets of parameters (blue crosses). Each set of parameters is place on graph in function of its concrete execution time (horizontal axis) and its actual memory cost (vertical axis). On the figure, red squares are sets of parameters of optimal points *i.e.* on the Pareto frontier of concrete execution results. Green circles are sets of parameters elected by exploration method *i.e.* on the Pareto frontier of exploration results. Those last points are placed depending on their concrete execution time and memory cost.

A set of parameters or optimal point is a set of parameters implying a weak memory consumption and/or a low execution time or a compromise between both. In other words, optimal points are placed in the bottom-left corner of graphs in Figure 6. Sets of parameters that are obtained by exploration are actually well placed in the bottom-left corner on graphs in Figure 6 and so, are close to optimal points. In order to evaluate the efficiency of the exploration method and the closeness of points, we calculated the distance between selected exploration points and optimal executed points.

The points in $\mathcal{P}$ are optimal if they are on the Pareto frontier. Points that are chosen in second position are points on the second Pareto frontier *i.e.* points on the

Table 4: Information about exploration execution for each application.

Application	Number of sets	Time for finding sets	Exploration time of all sets of parameters	Total time of execution	Speedup
FiveHB	186082	5 h 20 min 3 s	19 h 45 min 46 s	147 h 7 min 26 s	7.4
Pedagogic	22969	2 h 49 min 37 s	24 h 29 min 36 s	1024 h 31 min 17 s	41.8
Croissant	5684	2 h 27 min 41 s	29 h 53 min 17 s	289 h 16 min 2 s	9.7
Medical	5808	1 h 32 min 44 s	319 h 29 min 6 s	5872 h 9 min 56 s	18.4
TOTAL	220543	12 h 10 min 5 s	393 h 37 min 45 s	7333 h 4 min 42 s	18.6

Table 5: Exploration speedup for each application depending on explored schemes.

Scheme \ Application	FiveHB	Pedagogic	Croissant	Medical
FV	1.8	19.9	16.4	36.2
YASHE	13.3	61.0	73.6	137.5
F-NTRU	4.1	8.8	6.3	5.9

Pareto frontier of all executed sets of parameters $\mathcal{J}_2^{exec}$ minus points in the first Pareto frontier $\mathcal{P}$. And so on, we can define the n-th Pareto frontier, detailed below.

Definition 2 (n-th Pareto frontier)**.** *Let $\mathcal{P}_1$ be the set of points on the Pareto frontier of $\mathcal{J}_2^{exec}$. Let $\mathcal{P}_1 = FrontPareto(\mathcal{J}_2^{exec})$ with $FrontPareto(E)$ which returns the set of points on the Pareto frontier of the set E. The set of points on the n-th Pareto frontier is:*

$$\mathcal{P}_n = FrontPareto\left(\mathcal{J}_2^{exec} \setminus \bigcup_{i=1}^{n-1} \mathcal{P}_i\right). \tag{7}$$

From the index of Pareto frontier explained above, we calculated a ranking position for sets of parameters as defined as below.

Definition 3 (Ranking of a set of parameters)**.** *Let j be a set of parameters in $\mathcal{P}_n$ with $\mathcal{P}_n$ the set of points on the n-th Pareto frontier. The ranking position of j is:*

$$Card\left(\bigcup_{i=1}^{n-1} \mathcal{P}_i\right) + 1, \tag{8}$$

with $Card(E)$ the cardinal number of the set E.

This ranking position c allows to tell if a set of parameters j is the c-th chosen points on a total of $Card(\mathcal{J}_2^{exec})$ points. In other words, the set of parameters j is placed at p % in the total points ranking with:

$$p = \frac{c-1}{Card(\mathcal{J}_2^{exec})} \times 100 \tag{9}$$

Table 6 gives ranking of each exploration result. The ranking of sets of parameters is determined from concrete execution results (execution time and memory cost) and their index of Pareto frontier (Definition 2). This ranking is calculated, on the one hand, with equation 8 (index of ranking on the total of explored sets of parameters) and, on the other hand, with equation 9 (index of ranking in percentages, *e.g.* 0 % is the first ranking place and 100 % the last one).

Table 6 also informs about the distance between:

• execution time of a set of parameters and minimal execution time of optimal points,

• memory consumption of a set of parameters and minimal memory cost of optimal points.

For the four gathered application, 6 sets of parameters on 7 are placed in the six first percents of all explored sets of parameters. Among those sets of parameters, 4 are in the first three percents.

The results presented in this section show that the proposed exploration method provides a good solution for a designer to fastly determine, for a given application, the HE scheme and its input parameters providing interesting performances. Moreover, if required, the p-calibration step can be tuned to provide a configuration even closer from the actual optimal.

6. PAnTHErS usage

The tool PAnTHErS is freely available [37]. It now gathers modeled schemes, theoretical analysis method, p-calibration and exploration method. It is implemented in Python using Sage [36]. PAnTHErS aims at answering to two types of users' profiles: designer and HE expert.

This section presents goals of the two users and how they can use the tool PAnTHErS in order to accomplish their aims.

6.1. Designer

A designer wants to use HE in his future application. This user has no specific skills in cryptography. The application designer/programmer needs HE in order to secure data of his clients.

In order to choose the HE scheme, the designer uses PAnTHErS and its exploration method. The designer must define his application in the tool as a succession of HE basic functions. The designer can use the tool Cingulata [2] to convert his C++ application into a boolean circuit. Each boolean operation can be easily interpreted has HE basic functions (*e.g.* XOR corresponds to homomorphic addition). Finally, the designer launches the exploration

Table 6: Results returned by exploration for each application, ranking of exploration selected points and distance from those selected points to optimal results.

Application	Scheme	$\log(q)$	$\log(w)$	t	Time (s)			Memory cost (MiB)			Ranking		Distance with optimal point	
					Estim.	Exec.	Error	Estim.	Exec.	Error	Position	In %	Time (s)	Memory cost (MiB)
FiveHB	YASHE	46	10	2	0.13	0.13	10.8 %	2.77	2.5	10.9 %	1/186082	0	0	0
	YASHE	75	38	3	1.77	1.73	1.79 %	2.77	4.5	38.51 %	867/186082	0.47	1.61	2
Pedagogic	FV	297	23	2	10.42	10.26	1.62 %	99.37	93.8	5.94 %	119/22969	0.51	0.12	16.8
	FV	339	68	2	10.75	11.53	6.78 %	83.93	80.2	4.66 %	736/22969	3.2	1.39	3.2
Croissant	FV	243	81	2	12.91	13.15	1.82 %	39.26	39.8	1.37 %	309/5684	5.42	1.63	4.3
Medical	FV	393	44	2	167.25	175.23	4.55 %	269.55	265.5	1.53 %	1734/5808	29.84	24.27	30.9
	FV	437	88	2	171.02	167.45	2.14 %	237.29	240	1.13 %	133/5808	2.27	16.49	5.4

Table 7: Returned results after concrete execution of all sets of parameters of FV, YASHE and F-NTRU for the four studying applications.

Application	Scheme	$\log(q)$	$\log(w)$	t	Time (s)	Memory cost (MiB)
FiveHB	YASHE	46	10	2	0.13	2.5
Pedagogic	FV	307	35	2	10.32	83.6
	FV	310	35	2	10.14	84.1
	FV	312	40	2	10.39	79.9
	FV	313	38	2	10.26	83.6
	FV	314	40	2	10.37	81.1
	FV	315	40	2	10.38	80.3
	FV	315	43	2	10.37	80.6
	FV	320	46	2	10.51	79.4
	FV	328	56	2	11.37	77
	FV	331	57	2	10.87	79.3
	FV	337	57	3	10.95	78.3
	FV	338	63	2	11.08	77
	FV	342	70	2	11.01	77.4
Croissant	FV	253	89	2	11.80	35.5
	FV	265	94	2	11.51	37.2
	FV	276	99	2	11.68	36.5
Medical	FV	412	58	2	150.95	258.7
	FV	449	84	2	153.47	245.3
	FV	449	90	2	155.00	234.6
	FV	451	98	2	154.39	235.8
	FV	460	99	2	153.50	237.3

method which will return the most interesting scheme and input parameters.

PAnTHErS is accessible and easy to use (especially for exploration) thanks to its graphical interface, implemented with TKinter library [38].

6.2. HE expert

A HE expert aims at analyzing and/or improving his scheme. Indeed, the analysis of a scheme allows the expert to profile his internal functions in terms of execution time and memory consumption. Thus, the expert could optimize those functions, for example through parallelization or hardware acceleration. Moreover, HE expert could compare his scheme performances versus those of other schemes which are already available in PAnTHErS library.

In order to analyze and/or compare schemes, HE expert can use the PAnTHErS graphical interface to perform fine grained analyses. Moreover, tutorials [39] are available to help an expert who would like to add a new scheme into PAnTHErS.

Adding HE schemes consists in implementing classical algorithms (for instance those referred in section 2) but by using Atomic functions of PAnTHErS. Those functions act as substitutes for the implemented algorithm's elementary operations (*e.g.* addition, multiplication,...). Sage library [36] allows the user to create mathematics sets such as polynomial rings or fields. PAnTHErS supports user design of Atomic (or Specific) functions that are supported by Sage library. Thus, PAnTHErS easily allows the modeling of all kinds of HE schemes (based on LWE, NTRU, approximate-GCD) as well as algorithms from other application domains.

The HE expert will not find in PAnTHErS a turnkey solution, but, instead an open and extensible platform: new metrics, new analyses, new external modules can be added or plugged in, with potential reuse of existing ones.

A first new metric that would make sense is noise analysis to help calculating multiplicative depth of new schemes without even having the mathematical equation. This analysis could help HE experts in their scheme creation with function per function analysis.

7. Conclusion and future work

Through a modeling and analysis tool named PAnTHErS, a user (designer or HE expert) is now able to analyze a HE scheme or an application. PAnTHErS offers the possibility to launch several types of analysis: theoretical or concrete, for a scheme or an application, partial exploration (one parameter variation) or complete exploration (all parameters variation). The user chooses what kind of analysis best fits his needs.

The new functionality of PAnTHErS, the exploration method, allows, for a given application, to analyze every HE schemes and sets of input parameters implying the application multiplicative depth. Thus, it enables to elect scheme inducing the most interesting results (a low execution time, a weak memory consumption or any tradeoff in between).

This method has been tested on four applications, exhibiting different features. Exploration highlights YASHE [22] for *FiveHB* application and FV [1] for the three other applications. Comparing to application concrete executions with scheme and sets of parameters to explore, exploration of all sets of parameters is from 7.4 to 41.8 times faster (see Table 4 in section 5.2).

Among 7 sets of parameters selected by exploration (four gathered applications), 6 are close to optimal *i.e.* placed in the first six percents of sets of parameters ranking. The ranking is determined from concrete execution

results. Only one set of parameters has a deceptive ranking: it is out of the 20 first percents.

In the short term, future work will focus on refining calibration step on the exploration method by increasing the number of referencing points. In the long term, an interfacing between PAnTHErS and Cingulata [2] could be first created to easily add new applications into PAnTHErS in order to explore and/or analyze them. Secondly, execution of applications and schemes could benefit from hardware acceleration. For instance, Migliore et al. [40] created a co-design approach to speedup polynomials multiplication. PAnTHErS could take benefit from this work for faster evaluations.

References

[1] J. Fan, F. Vercauteren, Somewhat Practical Fully Homomorphic Encryption, IACR Cryptology ePrint Archive 2012 (2012) 144.

[2] CEA-LIST Crypto Team, Cingulata: open-source compiler toolchain, https://github.com/CEA-LIST/Cingulata/, commit fbc40d9e9948630047e4a60312c925594d14fc46, 2017.

[3] R. L. Rivest, L. Adleman, M. L. Dertouzos, On data banks and privacy homomorphisms, Foundations of secure computation 4 (1978) 169–180. URL: https://pdfs.semanticscholar.org/3c87/22737ef9f37b7a1da6ab81b54224a3c64f72.pdf.

[4] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (1978) 120–126. URL: http://dl.acm.org/citation.cfm?id=359342.

[5] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1999, pp. 223–238. URL: http://link.springer.com/chapter/10.1007/3-540-48910-X_16.

[6] C. Gentry, Fully homomorphic encryption using ideal lattices., in: STOC, 2009, pp. 169–178.

[7] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) Fully Homomorphic Encryption without Bootstrapping, in: ITCS, 2012.

[8] Z. Brakerski, Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2012, pp. 868–886. doi:10.1007/978-3-642-32009-5_50.

[9] Z. Brakerski, V. Vaikuntanathan, Efficient Fully Homomorphic Encryption from (Standard) LWE, FOCS 2011 (2011) 97–106. doi:10.1109/FOCS.2011.12.

[10] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, R. Sirdey, Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression, in: FSE, 2016.

[11] M. V. Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully Homomorphic Encryption over the Integers, in: Proc. EUROCRYPT, French Riviera, 2010, pp. 24–43. doi:10.1007/978-3-642-13190-5_2.

[12] J. Coron, A. Mandal, D. Naccache, M. Tibouchi, Fully Homomorphic Encryption over the Integers with Shorter Public Keys, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2011, pp. 487–504. doi:10.1007/978-3-642-22792-9_28.

[13] J. Coron, D. Naccache, M. Tibouchi, Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers, in: Proc. EUROCRYPT, Cambridge, UK, 2012, pp. 446–464. doi:10.1007/978-3-642-29011-4_27.

[14] J. H. Cheon, J. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, A. Yun, Batch Fully Homomorphic Encryption over the Integers, in: Proc. EUROCRYPT, Athens, Greece, 2013, pp. 315–335. doi:10.1007/978-3-642-38348-9_20.

[15] J. Coron, T. Lepoint, M. Tibouchi, Scale-Invariant Fully Homomorphic Encryption over the Integers, in: Proc. Public-Key Cryptography - PKC, Buenos Aires, Argentina, 2014, pp. 311–328. doi:10.1007/978-3-642-54631-0_18.

[16] R. Lindner, C. Peikert, Better Key Sizes (and Attacks) for LWE-Based Encryption, in: Proc. - CT-RSA 2011, San Francisco, CA, USA, 2011, pp. 319–339. doi:10.1007/978-3-642-19074-2_21.

[17] Z. Brakerski, V. Vaikuntanathan, Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2011, pp. 505–524. doi:10.1007/978-3-642-22792-9_29.

[18] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, in: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005, 2005, pp. 84–93. URL: http://doi.acm.org/10.1145/1060590.1060603. doi:10.1145/1060590.1060603.

[19] C. Gentry, A. Sahai, B. Waters, Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based, in: Proc. CRYPTO, Santa Barbara, CA, USA, 2013, pp. 75–92. doi:10.1007/978-3-642-40041-4_5.

[20] Z. Brakerski, V. Vaikuntanathan, Lattice-based FHE as secure as PKE, in: Proc. Innovations in Theoretical Computer Science, Princeton, NJ, USA, 2014, pp. 1–12. doi:10.1145/2554797.2554799.

[21] J. Hoffstein, J. Pipher, J. H. Silverman, NTRU: A ring-based public key cryptosystem, in: Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings, 1998, pp. 267–288. doi:10.1007/BFb0054868.

[22] J. W. Bos, K. E. Lauter, J. Loftus, M. Naehrig, Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme, in: Cryptography and Coding IMA, 2013.

[23] Y. Doröz, B. Sunar, Flattening NTRU for evaluation key free homomorphic encryption, IACR Cryptology ePrint Archive (2016).

[24] S. Halevi, V. Shoup, HElib - An implementation of homomorphic encryption, https://github.com/shaih/HElib, 2014.

[25] J.-S. Coron et al., An implementation of the DGHV fully homomorphic scheme, https://github.com/coron/fhe, 2012.

[26] C. A. Melchor, J. Barrier, S. Guelton, A. Guinet, M. Killijian, T. Lepoint, Nfllib: Ntt-based fast lattice library, in: Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings, 2016, pp. 341–356. doi:10.1007/978-3-319-29485-8_20.

[27] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, T. Lepoint, Nfllib: Ntt-based fast lattice library, in: Cryptographers' Track at the RSA Conference, Springer, 2016, pp. 341–356.

[28] CryptoExperts, FV-NFL : Library implementing the Fan-Vercauteren homomorphic encryption scheme, https://github.com/CryptoExperts/FV-NFLlib#fv-nfllib, 2016.

[29] C. Feron, V. Lapotre, L. Lagadec, PAnTHErS: A Prototyping and Analysis Tool for Homomorphic Encryption Schemes, in: Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT, Madrid, Spain, 2017, pp. 359–366.

[30] C. Feron, V. Lapotre, L. Lagadec, Fast evaluation of homomorphic encryption schemes based on ring-lwe, in: New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on, IEEE, 2018, pp. 1–5.

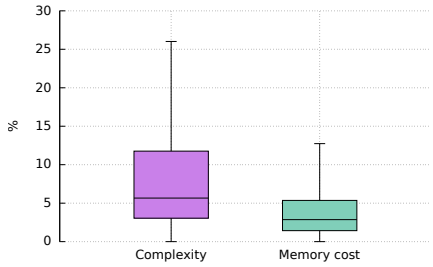
[31] M. Albrecht, S. Bai, L. Ducas, A subfield lattice attack on overstretched ntru assumptions, in: Annual Cryptology Conference, Springer, 2016, pp. 153–178.

[32] V. Pareto, Cours d'économie politique, volume 1, Librairie Droz, 1964.

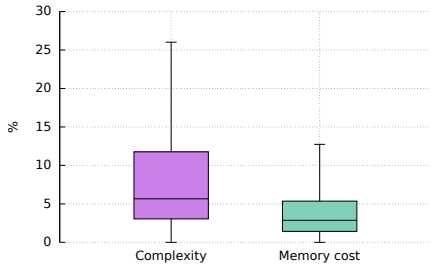
[33] U. o. C. Berkeley, Berkeley logic interchange format (blif), Oct Tools Distribution 2 (1992) 197–247.

[34] L. Lagadec, B. Pottier, O. Villellas-Guillen, An lut-based high level synthesis framework for reconfigurable architectures,

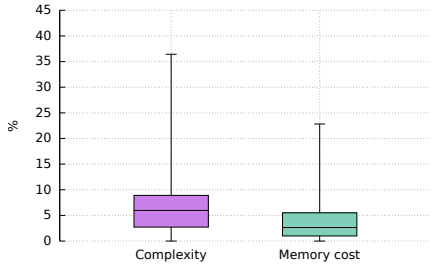
Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation (2003) 19–39.
[35] S. Carpov, T. H. Nguyen, R. Sirdey, G. Constantino, F. Martinelli, Practical privacy-preserving medical diagnosis using homomorphic encryption, in: Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on, IEEE, 2016, pp. 593–599.
[36] W. A. Stein, T. Abbott, M. Abshoff, SageMath, `http://www.sagemath.org/`, 2016.
[37] C. Feron, PAnTHErS (Prototyping and Analysis Tool for Homomorphic Encryption Schemes), `https://github.com/cferon/PAnTHErS`, 2018.
[38] J. W. Shipman, Tkinter 8.4 reference: a gui for python, New Mexico Tech Computer Center (2013).
[39] C. Feron, PAnTHErS: User Guide, Technical Report, ENSTA Bretagne ; Lab-STICC, 2018. URL: `https://hal.archives-ouvertes.fr/hal-01867913`.
[40] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using karatsuba algorithm, IEEE Trans. Computers 67 (2018) 335–347. doi:`10.1109/TC.2016.2645204`.



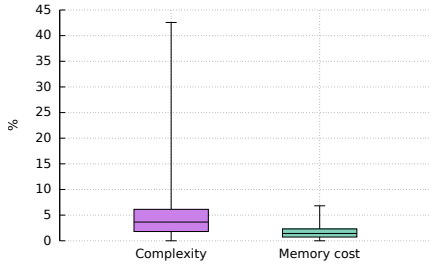
(a) *FiveHB* (98 %).



(b) *Pedagogic* (99 %).

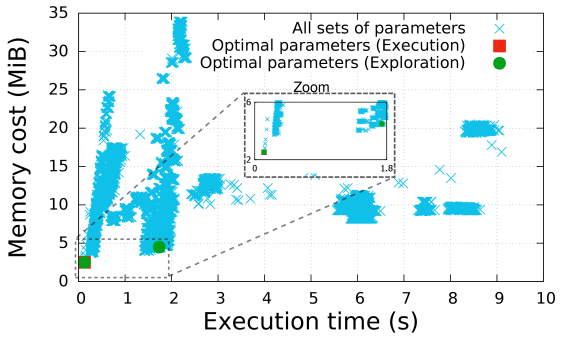


(c) *Croissant* (97 %).

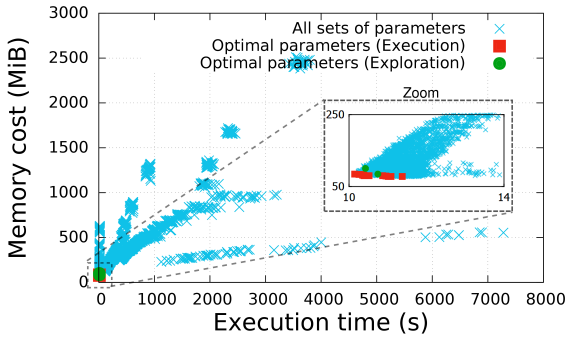


(d) *Medical* (99 %).

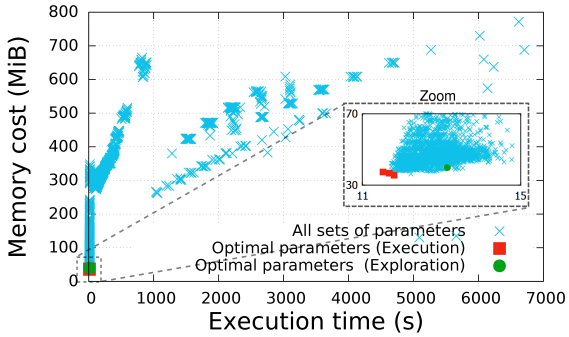
Figure 5: Boxplot of error rates representing differences between exploration results and concrete execution results, for each application and each calibrated theoretical analysis (complexity and memory cost).



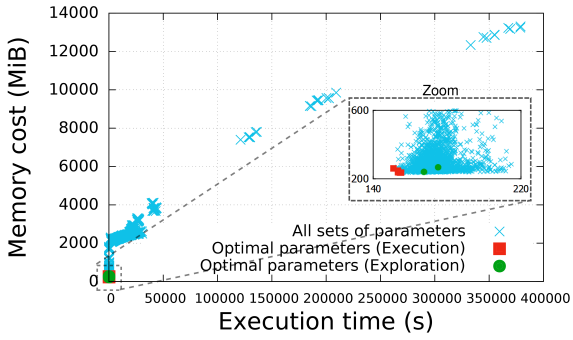
(a) *FiveHB*.



(b) *Pedagogic*.



(c) *Croissant*.



(d) *Medical*.

Figure 6: Identification of returned exploration points. Blue crosses are analysis results that are obtained after concrete execution of sets of parameters. Red squares are optimal points and green circles are exploration selected points.

6 Shuffling matériel pour l'algorithme de chiffrement AES dans un contexte IoT

Cette section présente les travaux de recherche réalisés dans le cadre de la thèse de Ghita Harcha [Harcha 2021] entre 2017 et 2021. Ces travaux s'inscrivent dans le cadre de la protection des architectures matérielles face aux attaques physiques. Plus particulièrement, nous nous sommes concentrés sur les attaques par canaux auxiliaires exploitant la consommation de puissance au sein d'un accélérateur matériel pour l'algorithme de chiffrement AES dans un contexte IoT.

6.1 Attaques par analyse de canaux auxiliaires

Certaines attaques physiques exploitent la capacité d'observation de grandeurs physiques pour déduire des informations sensibles manipulées par un circuit. En effet, des grandeurs physiques mesurables tels que le temps d'exécution, le rayonnement électromagnétique ou la consommation de puissance peuvent être corrélées aux traitements réalisés au sein d'un circuit. Durant les travaux de thèse de Mme Harcha, nous nous sommes particulièrement intéressés aux attaques dites DPA (*Differential Power Analysis*).

Les attaques DPA [Kocher 1999, Mangard 2008], exploitent la variation de consommation de puissance en fonction des données traitées à un instant donné. Par conséquent, lorsque qu'une donnée sensible est manipulée, la consommation de puissance résultante fournie une information précieuse qui peut être exploitée par un attaquant pour déduire la valeur (ou des caractéristiques) de la donnée sensible. Pour réaliser ce type d'attaque, la première étape consiste à faire l'acquisition, pour une cible donnée, de traces de consommation de puissance. Ces traces sont ensuite traitées (p. ex : filtrage, réduction du bruit, analyse fréquentielle, etc.) pour améliorer la performance des analyses statistiques réalisées afin d'extraire la donnée recherchée (c.-à-d. un secret). Ces analyses statistiques s'appuient alors sur un modèle de fuite et un distingueur.

Dans notre cas, un modèle de fuite se définit comme une fonction de prédiction théorique estimant une image de la consommation de puissance en considérant une valeur intermédiaire (ou une combinaison de valeurs intermédiaires) de l'algorithme considéré. On peut citer comme exemple les modèles de fuite basés sur la valeur d'un ou plusieurs bits ou sur le poids de Hamming d'une valeur intermédiaire, ou la distance de Hamming entre plusieurs valeurs intermédiaires. Un distingueur, quant à lui, est une métrique statistique permettant de distinguer la valeur secrète sur la base des observations. Le distingueur mesure alors le niveau de similarité entre le modèle de fuite pour une hypothèse de secret et la fuite existante dans les traces de consommation de puissance obtenues. Un des distingueurs les plus efficaces est la corrélation, menant à des attaques dites CPA (*Correlation Power Analysis*) [Brier 2004].

Dans le cadre des travaux de thèse de Mme Harcha, nous nous sommes intéressés au cas de l'algorithme de chiffrement AES.

6.2 AES - Advanced Encryptions Standard

AES [NIST 2001] est un algorithme de chiffrement symétrique largement utilisé pour la protection des communications numériques. Il est construit à partir d'un réseau de substitutions-permutations (*Subsitution-Permutation Network*) sur le modèle proposé par Shannon en 1949 pour définir un chiffrement idéal apportant une forte résistance aux attaques [Shannon 1949]. Il est constitué d'opérations de permutations qui ont pour rôle de générer de la confusion, c.-à-d. de créer une relation non triviale entre le texte clair et le texte chiffré. De plus, des opérations de substitutions permettent à chaque bit du texte chiffré d'être hautement non linéaire avec les bits du texte clair et de la clé.

Un bloc de données AES a une longueur fixe de 128 bits. Il existe trois longueurs de clé dans le standard : 128, 192 ou 256 bits. Selon, la longueur de la clé, les différentes opérations sont itérées respectivement 10, 12 ou 14 fois. Une itération est généralement appelée *ronde*. Durant

chaque ronde, quatre opérations sont réalisées pour mettre à jour un état interne présenté sous la forme d'une matrice de 4×4 octets. Avant la première ronde, une première opération nommée *AddRoundKey* est réalisée. Cette opération consiste à appliquer une opération XOR entre le texte clair et la clé initiale (on parle de *Key Whitening*). Puis, pour chaque ronde, les opérations suivantes sont réalisées :

— **ShiftRows** : il s'agit d'une opération de permutation où les octets de la i^{eme} ligne de la matrice d'états sont décalés d'un nombre i de colonne, avec $0 \leq i \leq 3$

— **SubBytes** : c'est une opération de substitution non-linéaire dont le but est la diffusion des bits au sein d'un même octet

— **MixColumns** : cette opération engendre une dépendance entre les octets de chaque colonne de la matrice d'état. Pour cela, la matrice d'état est multipliée avec une matrice constante, où chaque octet est considéré comme un élément de $GF(2^8)$. Cette opération est omise durant la dernière ronde

— **AddRoundKey** : lors de cette opération les octets de la matrice d'états se voient appliquer une opération XOR avec la clé de ronde courante, elle-même dérivée de la clé initiale via une opération dédiée de *Key scheduling*

6.3 Proposition d'un mécanisme de shuffling matériel

Dans le but de parer aux attaques par canaux auxiliaires exploitant la consommation de puissance, les contre-mesures existantes peuvent être classées dans deux catégories : les techniques de masquages (*masking*) et les techniques d'obscurcissement (*hiding*). Les travaux décrits dans cette section appartiennent à la seconde catégorie. Au niveau algorithmique, ce type de technique consiste généralement à injecter de l'aléa dans l'ordre des traitements (*shuffling*) quand cela est possible [Tunstall 2013, Pan 2009]. Dans le cas de l'algorithme de chiffrement AES, le shuffling des opérations peut être appliqué au sein de chacune des opérations d'AddRoundKey, de SubBytes et de ShiftRows au niveau octet (c.-à-d. chaque octet de la matrice d'état peut être traité dans un ordre aléatoire). Au sein de l'opération MixColumns, cela peut être réalisé au niveau colonne (c.-à-d. chaque colonne de la matrice d'état peut être traitée dans un ordre aléatoire).

Le shuffling est une technique qui a été largement étudiée pour des implémentations logicielles. En particulier une évaluation de l'efficacité et du coût de cette approche a été proposée dans [Veyrat-Charvillon 2012]. Cependant, en 2017, peu de travaux proposaient d'intégrer un tel mécanisme dans des architectures matérielles dédiées au chiffrement AES. Les travaux que nous avons proposés ont permis de développer et d'évaluer une architecture pour le chiffrement AES proposant un chemin de donnée de 8 bits et intégrant un mécanisme permettant la génération de permutations dans l'ordre des traitements. La figure 8 présente le concept d'architecture proposée.

La solution proposée s'appuie sur un PRNG (*pseudo-random number generator*) permettant de piloter un module de brassage. Ce dernier est construit à l'aide d'un réseau de permutation dont les entrées correspondent aux index de la matrice d'état de l'algorithme AES. Ce réseau étant piloté par un mot de contrôle issu du PRNG, la sortie du réseau de permutation fourni une séquence d'index permutés qui définit l'ordre des traitements réalisés par le module AES. Enfin, le contrôleur de l'architecture assure le pilotage de l'ensemble de l'architecture.

Dans les implémentations proposées, un module Trivium [Cannière 2008] a été utilisé pour réaliser la fonction PRNG. Deux réseaux de permutation ont principalement été évalués : le réseau de Benes [Beneš 1964] offrant 16! permutations possibles et le réseau Omega [Lawrie 1975] offrant $2^{(N \times log_2 N)/2}$ permutations possibles.

6.4 Principaux résultats

La figure 9 présente une synthèse des résultats obtenus durant ces travaux. Ici, 8 architectures sont comparées au regard de 2 métriques : la sécurité via le nombre d'octets de la clé secrète révélés

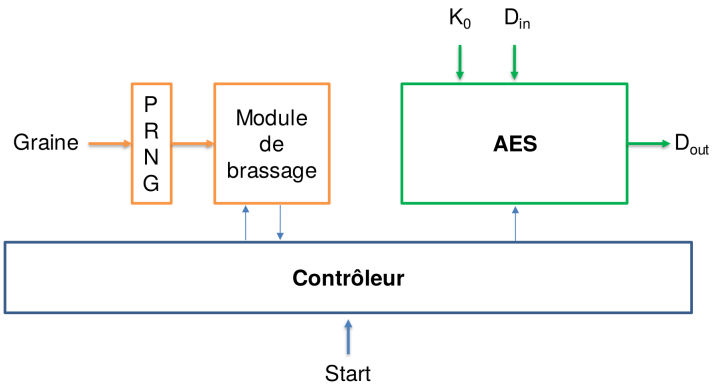


FIGURE 8 – Architecture AES intégrant un mécanisme de shuffling matériel

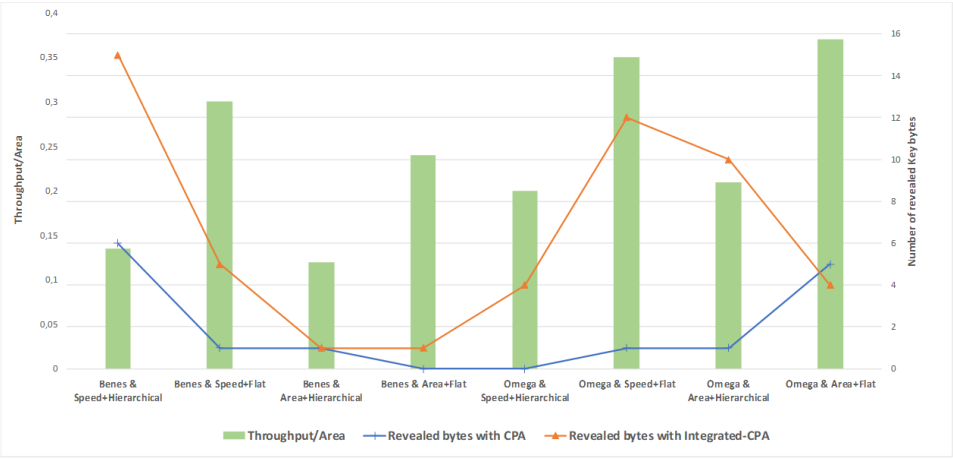


FIGURE 9 – Shuffling matériel - synthèse des résultats obtenus

via les deux attaques CPA considérées et le rapport débit/surface donnant une indication quant à l'efficacité de l'architecture. Ces 8 architectures diffèrent via le réseau de permutation utilisé et les options de synthèse choisies lors d'une implémentation sur cible FPGA Xilinx Spartan XC6SLX9 en utilisant l'outil Xilinx ISE 14.7. Les évaluations de sécurité ont été réalisées en s'appuyant sur la plateforme ChipWhisperer-Pro CW1200 kit [6] en collectant 1 million de traces pour chaque expérimentation.

Les résultats obtenus montrent une disparité importante dans la résistance aux attaques CPA considérées en fonction des options de synthèse choisies. On remarque que les solutions optimisées en surface permettent de réduire la sensibilité aux attaques. En effet, dans ces cas de figure, le rapport signal à bruit est réduit. Bien que moins efficace comparé à d'autres architectures, une solution intégrant un réseau de Benes optimisé en surface avec mise à plat de l'architecture semble être un choix judicieux. Elle combine un nombre important de permutations offert par le réseau Benes et une surface réduite réduisant le rapport signal à bruit. Il est important de noter que la technique de shuffling seule, ne permet pas de protéger efficacement les architectures matérielles faces aux attaques par canaux auxiliaires. Cependant, ce type d'approche peut être couplé à des contre-mesures de type masquage. Enfin, ces résultats montrent l'importance des options de synthèse dans la résistance à des attaques par canaux auxiliaires, appuyant le besoin croissant d'outils de synthèse *conscient* des enjeux de sécurité.

6. `https://www.newae.com/products/NAE-CW1200`

7 Conclusion

L'étude et l'implémentation de fonctions cryptographiques sont complexes et requièrent de nombreuses compétences afin d'aboutir à des solutions répondantes à des contraintes de performance et de sécurité. Les travaux de recherche réalisés en lien avec le chiffrement homomorphe ont permis à la fois d'appréhender les gains possibles en performances lors de l'utilisation d'une approche basée sur la co-conception logicielle / matérielle pour la réalisation des opérations coûteuses inhérentes à la cryptographie homomorphe et de faciliter la mise en oeuvre et l'évaluation de ces derniers via un outil dédié d'aide au prototypage.

Les travaux réalisés en lien avec l'implémentation de l'algorithme de chiffrement AES ont permis l'étude, le développement et l'évaluation d'une solution matérielle originale pour la génération de permutations permettant une mise en oeuvre efficace d'un mécanisme de shuffling. Ces travaux ont également mis en lumière le rôle que peut jouer un outil de synthèse dans la sensibilité d'une architecture à des attaques par canaux auxiliaires. La table 10 fournie quelques éléments de bilan des activités de cet axe de recherche.

TABLE 10 – bilan des activités pour l'axe de recherche "implémentation de fonctions cryptographiques"

Valorisation	3 revues et 6 communications internationales 1 communication nationale 1 communication par affiche et 1 logiciel
Collaborateurs	A. Tisserand (CNRS), C. Fontaine (CNRS) G. Gogniat (UBS), L. Lagadec (ENSTA Bretagne) P. Coussy (UBS), C. Chavet (UBS)
Doctorants	V. Migliore, C. Feron, G. Harcha
Ingénieur	C. Seguin
Financeurs	DGA, Brest Métropole, ENSTA Bretagne, Région Bretagne

7.1 Liste des publications associées à l'axe

7.1.1 Articles dans des revues internationales avec comité de lecture

[FER 2020a] C. Feron, V. Lapotre and L. Lagadec, **Automated Exploration of Homomorphic Encryption Scheme Input Parameters**, Journal of Information Security and Applications (JISA), 2020. Associate tool : PAnTHErS

[MIG 2018a] V. Migliore, M. Mendez Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, **Hardware/Software co-Design of an Accelerator for FV Homomorphic Encryption Scheme using Karatsuba Algorithm**, in IEEE Transactions on Computers, vol.37, Issue.3, pp.335-347, 2018.

[MIG 2017a] V. Migliore, C. Seguin, M. Mendez Real, V. Lapotre, A. Tisserand, C. Fontaine, G. Gogniat, **A High-Speed Accelerator for Homomorphic Encryption using the Karatsuba Algorithm**, ACM Transactions on Embedded Computing Systems (TECS), Volume 16 Issue 5s, 2017

7.1.2 Communications internationales avec comité de lecture et actes

[HAR 2020a] G. Harcha, V. Lapotre, C. Chavet, P. Coussy, **Toward Secured IoT Devices : a Shuffled 8-Bit AES Hardware Implementation**, in Proc. of 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020.

[FER 2018a] C. Feron, V. Lapotre and L. Lagadec, **Fast Evaluation of Homomorphic Encryption Schemes based on Ring-LWE**, in Proc. of 9th IFIP International Conference on New Technologies, Mobility & Security (NTMS), 2018.

[FER 2017a] C. Feron, V. Lapotre and L. Lagadec, **PAnTHErS : A Prototyping and Analysis Tool for Homomorphic Encryption Schemes**, 14th International Joint Conference on e-Business and Telecommunications - Volume 6 : SECRYPT, ICETE 2017, pages 359-366. 2017

[MIG 2016a] V. Migliore, M. Mendez Réal, V. Lapotre, A. Tisserand, C. Fontaine and G. Gogniat, **Fast polynomial arithmetic for Somewhat Homomorphic Encryption operations in hardware with Karatsuba algorithm**, in proc of the 2016 International Conference on Field-Programmable Technology (FPT '16), Jianguo Hotel, Xi'an, China, 2016.

[MIG 2015a] V. Migliore, M. Méndez Real, V. Lapotre, A. Tisserand, C. Fontaine and G. Gogniat, **Exploration of Polynomial Multiplication Algorithms for Homomorphic Encryption Schemes**, in Proc. of 2015 IEEE International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2015.

7.1.3 Communications internationales invitées sans actes

[BON 2017a] G. Bonnoron, C. Fontaine, G. Gogniat, V. Herbert, V. Lapotre, V. Migliore, A. Roux-Langlois, **Somewhat/Fully Homomorphic Encryption : Implementation Progresses and Challenges**, In : El Hajji S., Nitaj A., Souidi E. (eds), Codes, Cryptology and Information Security. C2SI 2017. Lecture Notes in Computer Science, vol 10194. Springer, Cham, 2017.

7.1.4 Communication nationale avec comité de lecture et actes

[MIG 2016b] V. Migliore, M. Méndez Real, V. Lapotre and G. Gogniat, **Algorithmes pour le chiffrement homomorphe**, Compas'2016.

7.1.5 Communication par affiche

[MIG 2015c] V. Migliore, M. Méndez Real, V. Lapotre, G. Gogniat, A. Tisserand, C. Fontaine, **Exploration of the best polynomial multiplication algorithm for homomorphic encryption schemes**, CHES 2015 Workshop on Cryptographic Hardware and Embedded Systems, 2015.

7.1.6 Logiciel libre

PAnTHErS Outils pour l'évaluation de schémas cryptographiques homomorphes. PAnTHErS fournit une estimation rapide du temps d'exécution et de l'empreinte mémoire pour

un jeu de paramètres représentatif du besoin applicatif. PAnTHErS fournit une solution d'exploration permettant à l'utilisateur de choisir un schéma de chiffrement et les paramètres associés en tenant compte des besoins applicatifs. L'outil est disponible via le lien suivant : `https://github.com/cferon/PAnTHErS`.

References

[Aguilar-Melchor 2016] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian et Tancrede Lepoint. *NFLlib : NTT-based fast lattice library.* In Cryptographers' Track at the RSA Conference, pages 341–356. Springer, 2016.

[Beneš 1964] Václad E Beneš. *Optimal rearrangeable multistage connecting networks.* Bell system technical journal, vol. 43, no. 4, pages 1641–1656, 1964.

[Bonnoron 2017] Guillaume Bonnoron, Caroline Fontaine, Guy Gogniat, Vincent Herbert, Vianney Lapôtre, Vincent Migliore et Adeline Roux-Langlois. *Somewhat/fully homomorphic encryption : Implementation progresses and challenges.* In International Conference on Codes, Cryptology, and Information Security, pages 68–82. Springer, 2017.

[Bos 2013] Joppe W Bos, Kristin Lauter, Jake Loftus et Michael Naehrig. *Improved security for a ring-based fully homomorphic encryption scheme.* In IMA International Conference on Cryptography and Coding, pages 45–64. Springer, 2013.

[Brakerski 2012] Zvika Brakerski. *Fully homomorphic encryption without modulus switching from classical GapSVP.* In In Advances in Cryptology - Crypto 2012, volume 7417 of Lecture, 2012.

[Brier 2004] Eric Brier, Christophe Clavier et Francis Olivier. *Correlation power analysis with a leakage model.* In International workshop on cryptographic hardware and embedded systems, pages 16–29. Springer, 2004.

[Cannière 2008] Christophe De Cannière et Bart Preneel. *Trivium.* In New stream cipher designs, pages 244–266. Springer, 2008.

[Doröz 2020] Yarkın Doröz et Berk Sunar. *Flattening NTRU for evaluation key free homomorphic encryption.* Journal of Mathematical Cryptology, vol. 14, no. 1, pages 66–83, 2020.

[Elgamal 1985] T. Elgamal. *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Transactions on Information Theory, vol. 31, no. 4, pages 469–472, 1985.

[Fan 2012] Junfeng Fan et Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption.* IACR Cryptol. ePrint Arch., page 144, 2012.

[Feron 2018] Cyrielle Feron. *PAnTHErS : un outil d'aide pour l'analyse et l'exploration d'algorithmes de chiffrement homomorphe.* Theses, ENSTA Bretagne - École nationale supérieure de techniques avancées Bretagne, Novembre 2018.

[Feron 2020] Cyrielle Feron, Loïc Lagadec et Vianney Lapôtre. *Automated exploration of homomorphic encryption scheme input parameters.* Journal of Information Security and Applications, vol. 55, page 102627, 2020.

[Fontaine 2007] C. Fontaine et F. Galand. *A Survey of Homomorphic Encryption for Nonspecialists.* EURASIP Journal on Information Security, 2007.

[Gentry 2009a] Craig Gentry. A fully homomorphic encryption scheme. Stanford university, 2009.

[Gentry 2009b] Craig Gentry. *Fully homomorphic encryption using ideal lattices.* In Proceedings of the forty-first annual ACM symposium on Theory of computing, pages 169–178, 2009.

[Harcha 2021] Ghita Harcha. *Introduction d'aléas dans les architectures matérielles pour une contribution à la sécurisation de chiffreurs AES dans un contexte IoT*. Theses, Université de Bretagne Sud, Juillet 2021.

[Karatsuba 1963] Anatolii Karatsuba. *Multiplication of multidigit numbers on automata*. In Soviet physics doklady, volume 7, pages 595–596, 1963.

[Khedr 2015] Alhassan Khedr, Glenn Gulak et Vinod Vaikuntanathan. *SHIELD : scalable homomorphic implementation of encrypted data-classifiers*. IEEE Transactions on Computers, vol. 65, no. 9, pages 2848–2858, 2015.

[Kocher 1999] Paul Kocher, Joshua Jaffe et Benjamin Jun. *Differential power analysis*. In Annual international cryptology conference, pages 388–397. Springer, 1999.

[Lawrie 1975] Duncan H. Lawrie. *Access and alignment of data in an array processor*. IEEE Transactions on Computers, vol. 100, no. 12, pages 1145–1155, 1975.

[Mangard 2008] Stefan Mangard, Elisabeth Oswald et Thomas Popp. Power analysis attacks : Revealing the secrets of smart cards, volume 31. Springer Science & Business Media, 2008.

[Migliore 2017] Vincent Migliore. *Cybersécurite matérielle et conception de composants dédiés au calcul homomorphe*. Theses, Université de Bretagne Sud, Septembre 2017.

[Migliore 2018] Vincent Migliore, Maria Méndez Real, Vianney Lapotre, Arnaud Tisserand, Caroline Fontaine et Guy Gogniat. *Hardware/Software Co-Design of an Accelerator for FV Homomorphic Encryption Scheme Using Karatsuba Algorithm*. IEEE Transactions on Computers, vol. 67, no. 3, pages 335–347, 2018.

[NIST 2001] NIST. *Advanced Encryption Standard*. NIST FIPS PUB 197, 2001.

[Paillier 1999] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In Jacques Stern, editeur, Advances in Cryptology — EUROCRYPT '99, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Pan 2009] Jing Pan, JI Den Hartog et Jiqiang Lu. *You cannot hide behind the mask : Power analysis on a provably secure s-box implementation*. In International Workshop on Information Security Applications, pages 178–192. Springer, 2009.

[Pollard 1971] John M Pollard. *The fast Fourier transform in a finite field*. Mathematics of computation, vol. 25, no. 114, pages 365–374, 1971.

[Regev 2009] Oded Regev. *On Lattices, Learning with Errors, Random Linear Codes, and Cryptography*. J. ACM, vol. 56, no. 6, sep 2009.

[Shannon 1949] Claude E Shannon. *Communication theory of secrecy systems*. The Bell system technical journal, vol. 28, no. 4, pages 656–715, 1949.

[Tunstall 2013] Michael Tunstall, Carolyn Whitnall et Elisabeth Oswald. *Masking tables—an underestimated security risk*. In International Workshop on Fast Software Encryption, pages 425–444. Springer, 2013.

[Veyrat-Charvillon 2012] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof et François-Xavier Standaert. *Shuffling against Side-Channel Attacks : A Comprehensive Study with Cautionary Note*. In Xiaoyun Wang et Kazue Sako, editeurs, Advances in Cryptology – ASIACRYPT 2012, pages 740–757, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Troisième partie

Contributions à l'axe : Protections face aux attaques exploitant la microarchitecture

Dans cette partie, je présente les travaux de recherche menés dans le cadre de l'étude, la conception et l'implémentation de protections face à des attaques exploitant les vulnérabilités de la microarchitecture des processeurs modernes. Ces travaux se sont particulièrement concentrés sur un sous-ensemble d'attaques exploitant des variations temporelles observables par un logiciel malveillant.

Dans une première section, une courte introduction à la problématique est proposée. Puis, une section est dédiée à la description des différents travaux réalisés dans le cadre de cet axe de recherche. Pour chacune de ces sections, je donne une brève introduction au sujet d'étude puis, je décris les principales contributions et enfin je fournis une sélection des principaux articles illustrant les travaux réalisés.

Les travaux décrits dans cet axe correspondent à deux projets de thèse de doctorat que j'ai co-encadrés et un projet de thèse que je co-encadre depuis octobre 2021 :

— Isolation spatiale contre les attaques par canaux auxiliaires exploitant les mémoires caches au sein des architectures many-core (2014 - 2017)

— Détection logicielle d'attaques exploitant des canaux auxiliaires exploitant les mémoires caches (2016 – 2019)

— Conception conjointe d'un processeur et du compilateur associé pour lutter contre les attaques par canaux auxiliaires exploitant les variations temporelles durant l'exécution de code (depuis 2021)

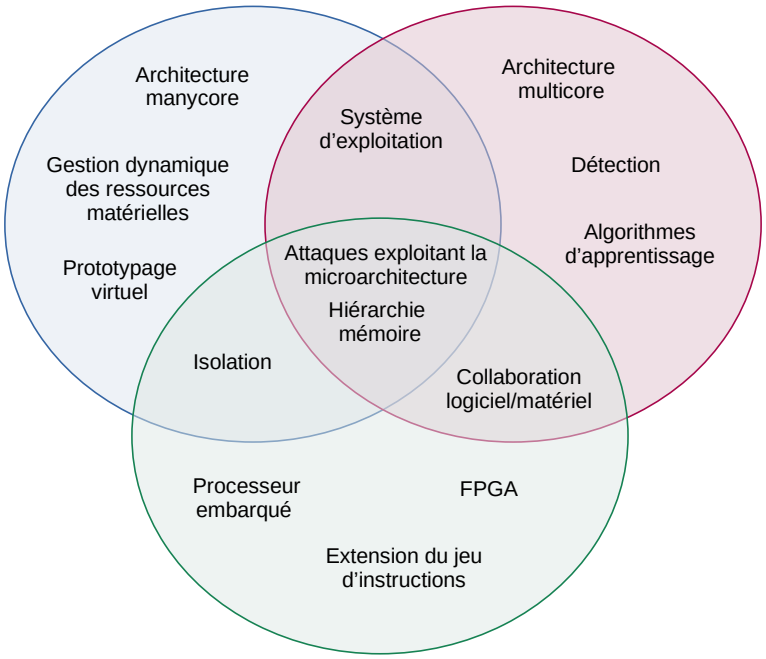


FIGURE 10 – Couverture thématique de l'axe protections face aux attaques exploitant la microarchitecture

La figure 10 présente des mots clés associés à chacun de ces travaux de thèse. Cette représentation permet de mettre en évidence les liens entre ces trois projets. Les différents travaux se focalisent principalement sur les attaques par canaux auxiliaires exploitant les éléments de la hiérarchie mémoire, en particulier les mémoires caches. Les deux premiers projets de thèse adressent

la problématique de la protection face à ces attaques du point de vue du système d'exploitation. Dans le premier cas, l'approche proposée vise à garantir l'isolation physique des applications en s'appuyant sur la gestion des ressources matérielles du système d'exploitation afin de créer des zones de sécurité au sein d'une architecture manycore. Dans le second cas, l'isolation physique ne pouvant être garantie dans l'architecture matérielle considérée, l'approche proposée vise à détecter un comportement malveillant trahissant une attaque en s'appuyant sur des algorithmes d'apprentissage. Pour cela, le matériel et le logiciel collaborent afin de réaliser une supervision de métriques issues des compteurs de performances du processeur. En cas de détection, le système d'exploitation pourra interrompre le processus à l'origine de l'attaque.

Le troisième projet de thèse, débuté en 2021, complémente les projets précédents en se focalisant sur le développement de protections matérielles au sein d'un processeur. La figure 10 permet d'identifier deux liens avec les projets précédents. Le premier lien concerne le renforcement de la collaboration entre le logiciel et le matériel pour mettre en oeuvre des protections efficientes et adaptées aux besoins applicatifs. Dans le cadre de ce projet, la chaine de compilation s'appuiera sur des extensions matérielles du processeur pour protéger, à la demande, les données sensibles manipulées par le processeur. La protection fournie s'appuiera, comme pour le second projet de thèse, sur un mécanisme d'isolation des données sensibles empêchant, par exemple, un attaquant de manipuler certaines lignes de mémoire cache.

Les attaques exploitant la microarchitecture sont une menace relativement récente. Ce type d'attaque à particulièrement été mis en lumière avec la publication des attaques spectre [Kocher 2019] et meltdown [Lipp 2018] en 2018. Néanmoins, de nombreux travaux sont réalisés dans ce domaine depuis le début des années 2000. Dans le cadre de mes travaux de recherche, je me suis principalement intéressé à un type particulier d'attaques : les attaques par canaux auxiliaires exploitant les variations temporelles au sein de la microarchitecture. La section suivante fournie une brève introduction à ces attaques.

8 Les attaques par canaux auxiliaires exploitant les variations temporelles

Un ensemble d'attaques par canaux auxiliaires exploitant la microarchitecture repose sur la capacité d'un logiciel malveillant à observer des variations temporelles lors de l'exécution d'un logiciel victime. Bien que les sources de ces variations soient multiples, elles sont en grande majorité liées aux optimisations et au partage des ressources au sein des architectures modernes. La figure 11, issue de [Ge 2018], présente un aperçu des ressources matérielles partagées au sein d'un processeur moderne. Dans la suite de la section, les principaux éléments exploités dans des attaques de l'état de l'art sont identifiés.

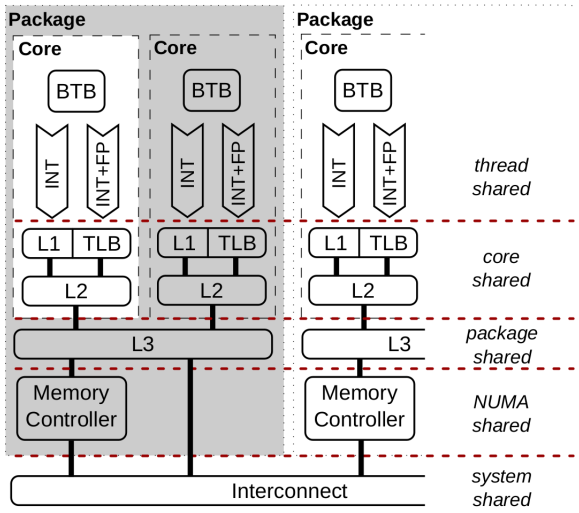


FIGURE 11 – Partage de ressources au sein d'une architecture de processeur moderne

Au sein d'un cœur de processeur, de nombreux éléments de la microarchitecture peuvent engendrer des fuites d'information. Le temps d'exécution d'une instruction peut varier en fonction des données manipulées. Par exemple, certains opérateurs arithmétiques pour la division réalisent une opération en un nombre de cycles variable en fonction de la valeur du dénominateur. On peut également citer l'exemple des instructions de branchement dont le temps d'exécution peut varier en fonction de l'issue, pris/non pris, du branchement. L'activation du SMT (*symmetric multithreading* engendre de la contention sur les ressources matérielles au sein d'un cœur. En effet, l'exécution concurrente au sein d'un même cœur de plusieurs *threads* mène à une compétition pour l'obtention de ressources partagées (c.-à-d. les unités fonctionnelles). Un thread malveillant est alors capable, par exemple en tentant d'accaparer une ressource, de détecter l'utilisation de cette même ressource par un autre thread (la victime). Le temps d'exécution du thread malveillant sera impacté lors de chaque utilisation de la ressource par le thread victime. Si l'utilisation de la ressource par la victime est liée à un secret, un attaquant sera en mesure de l'extraire. Le lecteur intéressé pourra approfondir ces sujets en se référant, par exemple, aux publications [Aciicmez 2007c] ou [Wang 2006].

Les différentes mémoires caches sont des ressources partagées particulièrement ciblées pour la réalisation d'attaques par canaux auxiliaires. Pour réaliser ces attaques, un attaquant s'appuie d'abord sur sa capacité à manipuler l'état de la mémoire cache partagée avec l'application victime. Il exploite alors les variations temporelles engendrées par la présence ou l'absence (*cache hit / cache miss*) d'une donnée pour déterminer les instructions où les données manipulées par la victime. Pour aller plus loin, le lecteur intéressé pourra se référer aux publications présentant trois des techniques d'attaque les plus populaires : Flush+Reload [Yarom 2014, Gullasch 2011], Prime+Probe [Osvik 2006, Percival 2005, Tromer 2010], Flush+Flush [Gruss 2016]. Les attaques visant les mémoires caches peuvent s'appliquer à différent niveau microarchitectural (cf. figure 11). Au sein d'un cœur, on peut citer les mémoires caches L1 et L2, mais également les mémoires stockant les translations d'adresses récentes (*Translation Lookaside Buffer* - TLB) [Hund 2013, Gras 2018] ou l'historique des issues de branchement (*Branch Target Buffer* - BTB) [Acıiçmez 2007b, Aciiçmez 2007a]. Au sein d'un *package*, un attaquant ciblera le dernier niveau de cache (L3 dans la figure 11). L'application malveillante s'exécute alors, parallèlement à la victime, sur un cœur différent. Cette position permet de réaliser une surveillance en temps réel des accès au dernier niveau de cache. Cela est particulièrement intéressant pour briser l'isolation entre deux machines virtuelles s'exécutant sur un même processeur [Yarom 2014]. Le lecteur trouvera dans l'article publié en 2020 [Mushtaq 2020b], issu de nos travaux dans le cadre de la thèse de Maria Mushtaq, une revue des attaques exploitant la hiérarchie de mémoires caches ciblant le système cryptographique RSA. En particulier, cet article présente les sources de fuites exploitées et les techniques d'attaque mises en oeuvre.

Il existe d'autres sources de fuites exploitables par un attaquant au sein de la microarchitecture. Afin d'obtenir une vision globale des possibilités, le lecteur pourra se référer à l'article [Ge 2018] qui introduit un ensemble de fuites, d'attaques et de contre-mesures de la littérature, et à l'article [Lou 2021] qui se focalise sur les travaux menés avec pour cible des applications cryptographiques.

Dans les sections suivantes, les travaux de recherche réalisés sur cette thématique sont présentés.

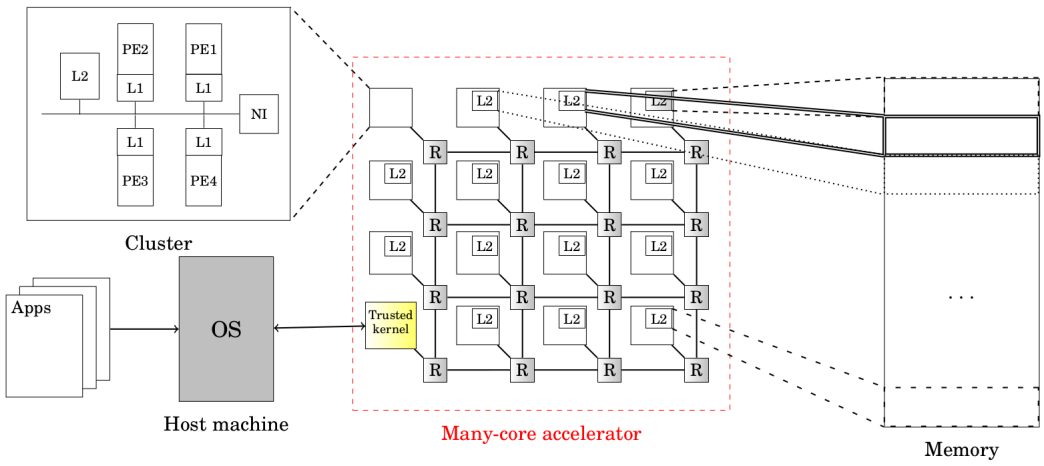


FIGURE 12 – Architecture TSAR utilisée comme accélérateur many-core

9 Isolation spatiale contre les attaques par canaux auxiliaires exploitant les mémoire caches au sein des architectures manycore

Les travaux présentés dans cette section ont été réalisés dans le cadre du projet collaboratif ANR TSUNAMY [7]. Ce projet s'intéresse à la protection des données manipulées au sein des architectures manycore dans un contexte d'informatique dans le nuage (*cloud computing*). Pour cela, diverses contributions sont proposées parmi lesquelles le développement d'une architecture manycore sécurisée intégrant des clusters hétérogènes cryptographiques, la mise en oeuvre d'un hyperviseur aveugle s'appuyant sur l'architecture matérielle pour assurer une isolation logique (au niveau logiciel) et physique (au niveau matériel) des machines virtuelles déployées et enfin le développement de stratégies permettant de répartir dynamiquement les applications sur l'architecture manycore tout en garantissant une protection face aux attaques par canaux auxiliaires exploitant les mémoires caches. Ce dernier élément a été le centre des travaux de thèse de Mme Maria Méndez Real que j'ai co-encadrée entre 2014 et 2017. La section suivante détaille le contexte de ces travaux.

9.1 Plateforme cible et modèle de menace

La plateforme manycore TSAR [8] est la plateforme cible des travaux réalisés dans le cadre du projet ANR TSUNAMY. La figure 12 extraite de [Real 2018] présente le contexte d'utilisation de l'architecture many-core TSAR dans le contexte de nos travaux. L'architecture TSAR est utilisée en tant qu'accélérateur matériel piloté par une machine hôte. Elle se compose de *clusters* interconnectés via un réseau sur puce à maillage simple à deux dimensions (*2D-mesh*). Chaque cluster est composé de quatre éléments de traitement (*Processing elements - PE*) associés à une mémoire cache privée (*L1*), une mémoire partagée (*L2*) et une interface de communication réseau connectant le cluster au réseau sur puce. Il est important de noter que dans la version originale de l'architecture, la mémoire L2 est partagée par tous les clusters. De plus, l'espace d'adressage physique de la mémoire principale est statiquement segmenté au regard du nombre de clusters. Chaque segment est statiquement assigné à une mémoire cache de niveau L2. Cette spécificité permet de mettre en oeuvre une politique de déploiement des données en prenant en considération la localité spatiale. Un cluster de l'architecture est dédié à l'exécution du système d'exploitation ALMOS [9] [Almaless 2014]. ALMOS est alors en charge de la gestion des ressources

7. `https://anr.fr/Projet-ANR-13-INSE-0002`
8. `https://www-soc.lip6.fr/trac/tsar`
9. Advanced Locality Management Operating System

matérielles et du déploiement des applications.

Dans ces travaux, l'architecture matérielle TSAR et le système d'exploitation ALMOS (*Trusted kernel* dans la figure 12) sont considérés de confiance. Les communications avec la machine hôte et les périphériques externes sont protégées. Les attaques physiques ne sont pas intégrées au modèle de menace considéré. Toutes les applications déployées sur l'architecture sont considérées comme potentiellement malveillantes. Le modèle de menace considéré inclut une éventuelle collaboration entre plusieurs applications pour réaliser une attaque contre le système ou une application en cours d'exécution. Les travaux présentés dans cette section se focalisent sur les attaques par canaux auxiliaires exploitant les mémoires caches au sein d'une architecture many-core. Une solution de protection basée sur la création de zones de sécurité (*secure zones*) lors du déploiement d'applications a été étudiée, développée et évaluée.

9.2 Stratégies de déploiement d'applications face aux attaques par canaux auxiliaires exploitant les mémoires caches

L'approche proposée s'appuie sur la notion d'isolation spatiale. En effet, puisque les attaques par canaux auxiliaires exploitant les mémoires caches s'appuient sur le partage de ressources, la solution proposée doit permettre d'empêcher ce partage. Pour cela, lors du déploiement d'une application, cette dernière peut être déclarée "sensible" par l'utilisateur ou le système hôte. Le système d'exploitation ALMOS, en charge de la gestion des ressources de la plateforme TSAR, est étendu pour permettre la création de zone de sécurité regroupant un ensemble de ressources matérielles (c.-à-d. des clusters) dédiées temporairement à l'exécution de cette application. Dans le but de maximiser les performances, il est préférable de construire des zones de sécurité en utilisant des clusters contigus. Ces zones sont créées et gérées dynamiquement par le système au regard des besoins de chaque application à isoler. De cette manière, le partage des mémoires caches avec une application isolée est évité. Lorsqu'une application isolée a été exécutée, la mémoire à l'intérieur de sa zone de sécurité est effacée afin d'éviter toute fuite d'information.

La figure 13 présente le principe d'isolation spatial dans un scénario où six applications sont exécutées sur la plateforme. Parmi ces six applications, deux sont déclarées sensibles et seront donc isolées. Quatre instants dans le temps sont représentés ($t0 < t1 < t2 < t3$). À *t0*, Aucune application n'est déployée. À *t1*, 3 applications, dont une application sensible ont été déployées. À cet instant, les trois applications ne requièrent pas plus d'un cluster pour s'exécuter. À *t2* les applications déployées à *t1* ont augmenté leurs besoins en ressources et deux nouvelles applications ont été déployées. En raison de la charge croissante sur la plateforme les applications non isolées partagent alors des ressources au sein des clusters. Cependant, ce partage n'est pas mis en oeuvre pour les applications isolées. Avant le déploiement d'une application sensible, le logiciel en charge de la gestion des ressources s'assure de la disponibilité d'un ensemble de clusters dont aura besoin l'application. À *t3*, lorsqu'une application isolée termine son exécution, les ressources qui lui était réservées sont libérées. Ces ressources sont déclarées à nouveau disponibles et peuvent être utilisées par d'autres applications. Afin d'intégrer les nouveaux services nécessaires à la gestion dynamiquement des zones de sécurité, les services du système d'exploitation ALMOS ont été étendus. Il est important de noter que le fait de dédier des clusters entiers aux applications isolées peut entraîner une sous-utilisation des ressources matérielles et, par conséquent, une dégradation des performances. C'est pourquoi, différentes stratégies de déploiement ont été évaluées :

1. déploiement au sein d'une zone de sécurité statique
2. déploiement au sein d'une zone de sécurité dynamique
3. déploiement au sein d'une zone de sécurité hybride

Dans l'approche statique, des ressources sont réservées avant le déploiement de l'application. La quantité de ressources à réserver est alors fixée par l'utilisateur ou à l'aide d'une phase de profilage de l'application. L'application sera déployée uniquement si la réservation de toutes les ressources demandées est possible. Dans le cas contraire, le déploiement de l'application est

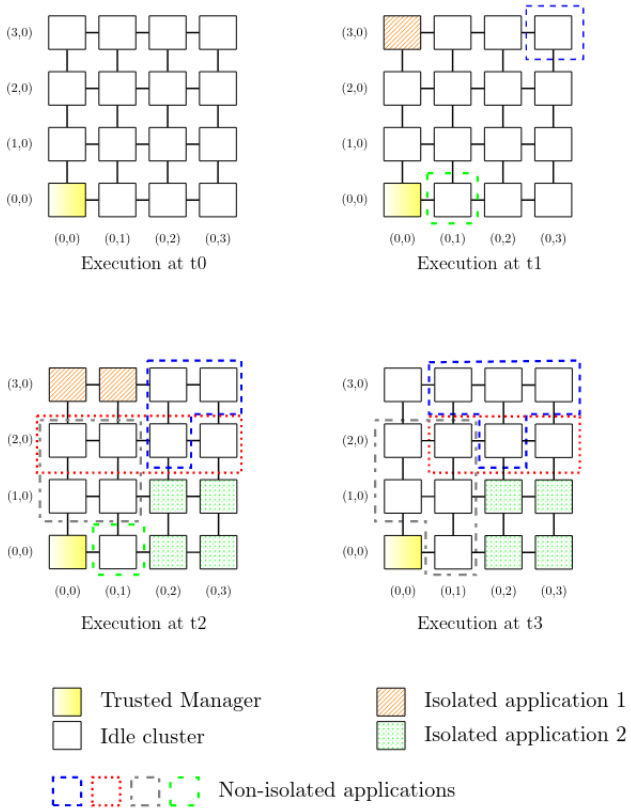


FIGURE 13 – Principe d'isolation spatiale au sein d'une architecture many-core

retardé dans l'attente de la libération des ressources nécessaires à son isolation. La principale limite de cette approche est liée au risque de sous-utilisation des ressources de la plateforme. En effet, si les besoins en ressources de l'application varient durant son exécution, ces derniers ne seront pas pris en compte et des ressources réservées ne seront pas toujours utilisées.

Pour palier à cette limite, une approche dynamique a été proposée. Dans cette dernière, la réservation et la libération des ressources au sein de la zone de sécurité se fait à l'exécution au regard des besoins applicatifs et de la charge sur la plateforme. Lors du déploiement d'une application isolée, une zone de sécurité constituée d'un unique cluster est créée. Des clusters supplémentaires sont ajoutés à la zone de sécurité lors de l'ordonnancement de nouvelles tâches ou de nouvelles allocations mémoires uniquement si les ressources actuellement disponibles dans la zone de sécurité sont insuffisantes. Lorsqu'un cluster supplémentaire est nécessaire, le système recherche un cluster libre contigu à la zone de sécurité actuelle. Si aucun cluster contigu n'est libre, la nouvelle tâche ou l'allocation mémoire ayant déclenchée la recherche d'un cluster libre est suspendue en attendant qu'une ressource se libère au sein de la zone de sécurité ou qu'un cluster contigu à la zone de sécurité soit libéré par une autre application. Cette stratégie permet de déployer une application sans attendre qu'un nombre spécifique de clusters soient disponibles pour la création d'une zone de sécurité. Elle permet également une gestion plus fine des ressources dans le but d'améliorer le taux d'utilisation des ressources. Cependant, en commençant le déploiement d'une application sur un unique cluster sans garanti de pouvoir augmenter la taille de la zone de sécurité, il existe un risque important de pénaliser les performances des applications faisant appel au mécanisme d'isolation.

Un compromis peut alors être trouvé avec une stratégie hybride permettant de créer une zone de sécurité incluant un minimum de ressources lors du déploiement de l'application. Les ressources constituant cette zone de sécurité initiale, ne peuvent alors pas être libérées durant l'exécution de l'application isolée. Cependant, à la suite du déploiement, l'approche dynamique

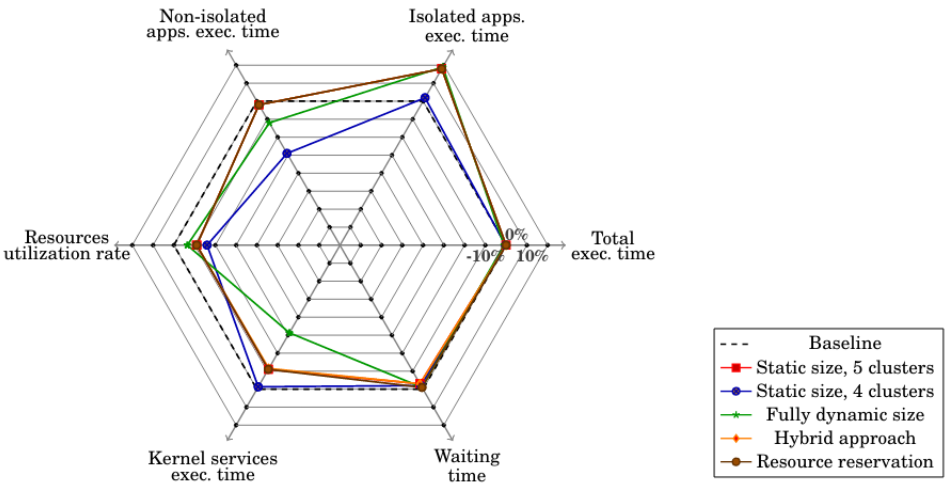


FIGURE 14 – Comparaisons des différentes stratégies pour le scénario **a**. les valeurs sont présentées en terme de coût (en %) comparés au déploiement de référence "*baseline*". Pour toutes les métriques, les valeurs proches du centre du graphique sont favorables

précédente est adoptée pour la gestion de ressources supplémentaires ajoutées à la zone de sécurité initiale. Une variante de cette stratégie a été proposée durant nos travaux. Elle consiste, en plus de la réservation des ressources pour la création de la zone de sécurité initiale, à pré-réserver des ressources contiguës à cette zone mais qui ne sont actuellement pas disponibles. Ces ressources pourront alors être allouées à la zone de sécurité dès qu'elles seront libérées par l'application les utilisant.

9.3 Principaux résultats

Les stratégies présentées dans la sous-section précédentes ont été évaluées via une solution de prototypage virtuel s'appuyant sur une version étendue de MPSoCSim présenté dans [Méndez Real 2016]. Pour cela, une plateforme composée de 16 clusters a été modélisée de façon à représenter au mieux l'architecture TSAR. 1 cluster est réservé pour le management de la plateforme (déploiement et ordonnancement des applications, réservation et libération des ressources, management des zones de sécurité) et 15 autres clusters, incluant 4 processeurs chacun, sont utilisés pour l'exécution des applications.

L'évaluation des stratégies s'est basée sur 3 scénarios détaillés ci-dessous. Pour chaque scénario, les applications considérées sont composées de 17 tâches idéalement parallélisables sur 5 clusters de l'architecture.

— **scénario a**. Dans ce scénario, 5 applications dont une nécessitant une isolation, sont exécutées. Une priorité élevée est accordée à l'application isolée. Celle-ci est donc ordonnancée avant les autres en considérant une charge nulle sur la plateforme au démarrage de l'expérimentation.

— **scénario b**. Ce scénario reprend le scénario précédent en donnant une priorité moyenne (4ème plus faible priorité parmi les 5 applications) à l'application nécessitant une isolation. Ce scénario permet de prendre en compte l'impact sur les performances dû à l'état de charge de la plateforme lors de la création d'une zone de zone de sécurité.

— **scénario c**. Dans ce scénario, 3 applications sur 5 nécessitent une isolation. Ces applications possèdent un niveau de priorité de 1, 3 et 5 respectivement. Ce scénario permet d'étudier l'impact du nombre d'applications isolées sur les performances.

Les figures 14, 15 et 16 extraites de [Real 2018], permettent de visualiser et d'analyser les stratégies de déploiement proposées pour chacun des scénarios. Dans ces figures, les courbes sont

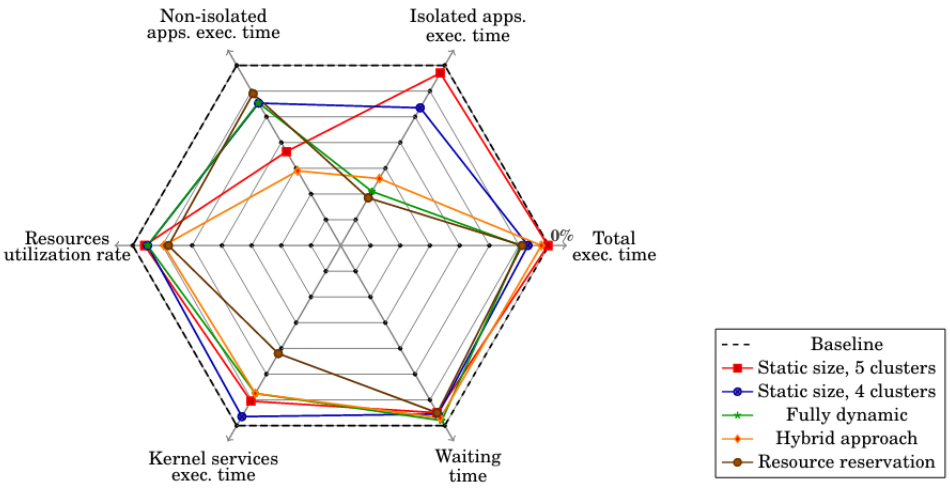


FIGURE 15 – Comparaisons des différentes stratégies pour le scénario **b**. les valeurs sont présentées en terme de coût (en %) comparés au déploiement de référence "*baseline*". Pour toutes les métriques, les valeurs proches du centre du graphique sont favorables

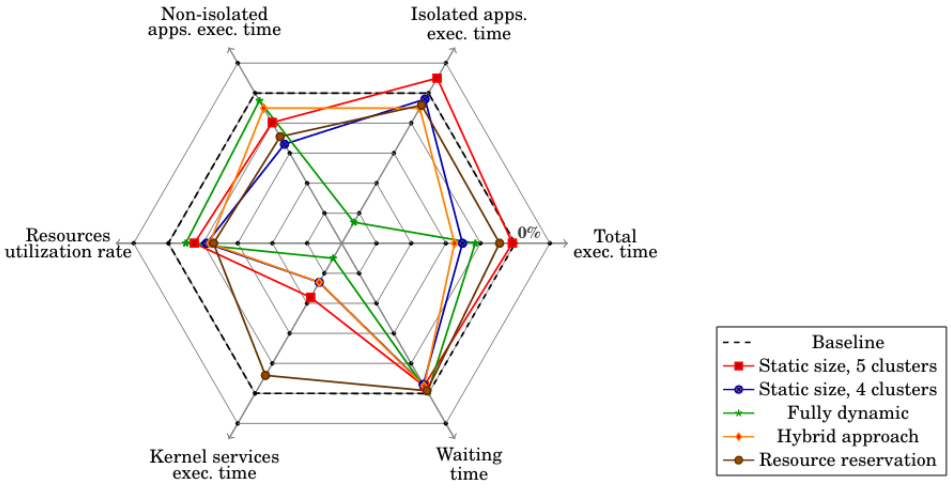


FIGURE 16 – Comparaisons des différentes stratégies pour le scénario **c**. les valeurs sont présentées en terme de coût (en %) comparés au déploiement de référence "*baseline*". Pour toutes les métriques, les valeurs proches du centre du graphique sont favorables

TABLE 11 – Intérêt des stratégies proposées en fonction de conditions d'exécution

scénarios d'exécution	zones de sécurité statiques		zones de sécurité dynamiques	zones de sécurité hybrides	
	taille optimale	taille limitée		taille minimum garantie	pré-réservation
les applications isolées sont caractérisées	✔	✔	-	✔	✔
La performance des applications isolées est critique	✔	✔	✖	✔	✖
La performance des applications isolées doit être favorisée	✖	-	✔	✔	✖
Le taux d'utilisation des ressources doit être optimisé	-	-	✔	-	-
La charge est faible sur la plateforme	✔	✖	-	-	-
L'activité des services de l'OS doit être optimisée	-	-	✖	-	-

associées à l'implémentions des stratégies de déploiement suivantes :

— ***Baseline***. Cette stratégie est utilisée comme référence pour l'analyse des résultats. Elle correspond à l'exécution du scénario sans aucune isolation des applications

— ***Static size 5 clusters***. Cette stratégie correspond au déploiement des applications isolées au sein d'une zone de sécurité statique dont le nombre de clusters est fixé à 5. Ce nombre de cluster permet l'exécution parallèle des 17 tâches des applications isolées

— ***Static size 4 clusters***. Cette stratégie correspond au déploiement des applications isolées au sein d'une zone de sécurité statique dont le nombre de clusters est fixé à 4

— ***Fully dynamic***. Cette stratégie correspond au déploiement des applications isolées au sein d'une zone de sécurité dynamique

— ***Hybrid approach***. Cette stratégie correspond au déploiement des applications isolées au sein d'une zone de sécurité hybride incluant un minimum de ressources lors du déploiement de l'application. Dans ce cas d'étude, le nombre minimum de clusters lors de la création d'une zone de sécurité est fixé à 2

— ***Ressource reservation***. Cette stratégie correspond à la stratégie précédente utilisant également la pré-réservation de ressources

L'analyse de ces trois figures regroupant des métriques de performance temporelle (applicatifs et système d'exploitation) ainsi que le taux d'utilisation des ressources, ne permet pas de faire ressortir une stratégie qui conviendrait à tous les cas de figure. En effet, on observe que pour chaque scénario, une stratégie peut grandement améliorer une métrique au détriment d'une autre. Par exemple, dans la figure 15, on observe que la stratégie *Ressource reservation* permet d'optimiser le temps d'exécution des applications isolées et des services du système d'exploitation, mais pénalise les applications non isolées en comparaison avec les autres stratégies. Ce même raisonnement peut être mené pour la stratégie *Fully dynamic* dans le cas du scénario **c** présenté en figure 16. En analysant ces résultats, nous avons été en mesure de proposer des conditions d'exécution permettant de discriminer les stratégies proposées. Cette analyse est résumée dans la table 11. Cette table indique, pour chaque scénario d'exécution, si la stratégie de déploiement est souhaitable (coche), non particulièrement souhaitable (tiret) ou non souhaitable (croix). Par exemple, si l'on considère que les applications isolées sont préalablement profilées pour en extraire leurs caractéristiques clés comme le niveau de parallélisme ou la consommation mémoire, toutes les stratégies exceptée celle s'appuyant sur un déploiement dynamique sont pertinentes. En effet, le nombre de ressources nécessaires à l'application étant connu, les approches statiques permettront de créer une zone de sécurité adaptée au besoin. Les stratégies hybrides bénéficieront également de cette connaissance. À contrario, la stratégie dynamique ne tiendra pas compte de cette information, pénalisant les performances temporelles de l'application déployée.

Les contributions présentées dans cette section sont détaillées dans l'article ci-dessous publié en 2018 dans la revue ACM Transactions on Embedded Computing Systems [Real 2018].

Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators

MARIA MÉNDEZ REAL, University of Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France
PHILIPP WEHNER, Ruhr-University Bochum, Germany
VIANNEY LAPOTRE, University of Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France
DIANA GÖHRINGER, Technische Universitatet Dresden, Germany
GUY GOGNIAT, University of Bretagne-Sud, UMR 6285, Lab-STICC, F-56100 Lorient, France

Current cache Side-Channel Attacks (SCAs) countermeasures have not been designed for many-core architectures and need to be revisited in order to be practical for these new technologies. Spatial isolation of resources for sensitive applications has been proposed taking advantage of the large number of resources offered by these architectures. This solution avoids cache sharing with sensitive processes. Consequently, their cache activity cannot be monitored and cache SCAs cannot be performed. This work focuses on the implementation of this technique in order to minimize the induced performance overhead. Different strategies for the management of isolated secure zones are implemented and compared.

CCS Concepts: • **Security and privacy → Security services**; **Trusted computing**; *Side-channel analysis and countermeasures*;

Additional Key Words and Phrases: Cache-based SCAs, many-core accelerator

ACM Reference format:
Maria Méndez Real, Philipp Wehner, Vianney Lapotre, Diana Göhringer, and Guy Gogniat. 2018. Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators. *ACM Trans. Embed. Comput. Syst.* 17, 2, Article 55 (February 2018), 31 pages.
https://doi.org/10.1145/3168383

1 INTRODUCTION

Side-Channel Attacks (SCAs) (Ge et al. 2016) allow an attacker, who has no direct access to critical data, to analyze indirect or side-channel information during or after the execution of a sensitive application (e.g., a cryptographic algorithm) in order to deduce the sensitive application behavior or critical information such as a cryptographic key. Depending on the side-channel information to exploit, the attacker may or may not require physical access to the system. We focus on logical cache-based SCAs, which do not require any physical access and which see the memory cache as

The work presented in this article was realized in the frame of the TSUNAMY project (TSUNAMY 2016) number ANR-13-INSE-0002-02 supported by the French Agence Nationale de la Recherche.
Authors' addresses: M. M. Real, UMR CNRS 6164 Institut d'Electronique et de, Télécommunications de Rennes (IETR), Polytech, Nantes, University of Nantes, Rue C. Pauc, BP 50609, 44306 Nantes, Cedex 3, France; email: maria.mendez@univ-nantes.fr; P. Wehner, Ruhr University: Universitätsstraße 150, 44801 Bochum, Allemagne; email: philipp.wehner@rub.de; V. Lapotre and G. Gogniat, University of Bretagne-Sud, Lab-STICC address: Rue de Saint-Maudé, 56100 Lorient, France; emails: {vianney.lapotre, guy.gogniat}@univ-ubs.fr; D. Göhringer, Technische Universitatet Dresden: 01069 Dresden, Allemagne; email: diana.goehringer@tu-dresden.de.

https://doi.org/10.1145/3168383

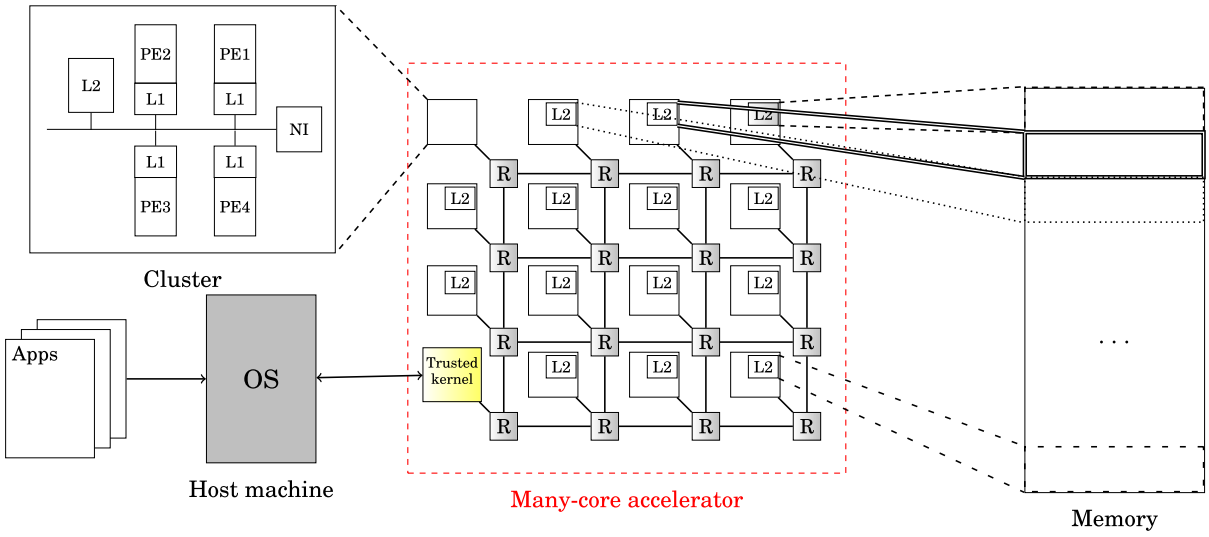


Fig. 1. Overview of the considered system.

the source of leakage. The cache is indeed a resource that several concurrent processes, sensitive and potentially malicious, compete for. When shared with an adversary, the latter can extract some information about the victim's activity that can be used to perform cryptanalysis. These attacks can be performed at different granularities; within a single core when the victim and attacker applications execute on the same core and share the level 1 (L1) cache (Percival 2005), as well as across cores when the victim and attacker applications execute on different cores but share the Last-Level Cache (LLC). These attacks have been proved practical as well in a cloud environment when the victim and the attacker applications execute on different Virtual Machines (VMs) logically isolated, since applications across VMs are still sharing the LLC (Yarom and Falkner 2014; Liu et al. 2015; Irazoqui et al. 2015). These attacks can steal sensitive information from systems implementing logically isolated execution environments (Evtyushkin et al. 2016). Recently, these attacks have been extended to multiprocessor and Network-on-Chip (NoC)-based systems (Irazoqui et al. 2016; Reinbrecht et al. 2016b). We focus on clustered NoC-based multi- and many-core architectures (Figure 1). In fact, the latter architectures have recently emerged as they provide massive parallelism and high performance allowing a wide number of applications, from different sources and with different levels of trust, to be executed in parallel sharing physical resources (computation, memory, and communication infrastructure). One example is the automotive field, in which the vehicle computer is indeed responsible for handling multimedia applications (user interface, car navigation, entertainment such as radio, Internet, etc.), safety-critical vehicle applications (airbag and braking system, for instance), vehicle communication applications (in a connected car context), as well as vehicle and engine management applications. In this context, multi- and many-core systems could cope with the performance requirements (Trung-Dung et al. 2016; Burgio et al. 2016). However, the cohabitation of these different applications sharing physical resources makes mandatory the integration of security mechanisms. In particular, sensitive applications sharing caches with potentially malicious ones are vulnerable to logical cache-based SCAs and can be attacked. A large amount of works aiming at countering these attacks have been proposed (Section 2.2). However, most of them have not been designed for recently emerged many-core architectures. These solutions require to be evaluated and/or revisited in order to be practical and efficient on multi- and many-core architectures.

A countermeasure taking advantage of the large number of resources in many-core architectures has been proposed in Méndez Real et al. (2016a). The aim is to prevent cache sharing with a sensitive application by temporarily dedicating part of the physical resources in order to execute

it within an isolated environment (*secure zone*). All the resources within the secure zone are dedicated to this single isolated application preventing it from cache sharing. Consequently, its cache activity can no longer be monitored and cache-based SCAs can no longer be performed. In the meanwhile, non-isolated applications execute normally and still share resources.

This article is an extension of the work of Méndez Real et al. (2016a). It particularly focuses on its implementation in order to study and reduce the induced performance overhead. In this work, we extend the work of Méndez Real et al. (2016a) with different deployment strategies for the dynamic creation and management of the secure zones. These strategies are evaluated on a Tera-Scale ARchitecture (TSAR)-like architecture (TSAR 2014) and are compared in terms of induced performance overhead according to several performance metrics. Results show that hybrid strategies between completely static and dynamic secure zone size approaches offer more flexibility and reduce the induced performance overhead on the total execution time compared to fully dynamic approaches.

The new contributions presented in this article are:

- The implementation of a set of services on a manager kernel based on the Advanced Locality Management Operating System (ALMOS) Operating System (OS) kernel (Almaless 2014) in order to dynamically control the deployment of applications.
- The extension of the manager kernel in order to integrate the secure-enable mechanisms proposed in Méndez Real et al. (2016a), guaranteeing the physically isolated execution of each sensitive application.
- The proposal of different secure zones deployment strategies including a static and dynamic secure zone size, as well as hybrid deployment strategies targeting a minimum guaranteed performance and resources reservation.
- The implementation and evaluation of the proposed strategies on a TSAR-like many-core architecture through virtual prototyping.
- The comparison of the gathered results regarding performance metrics including the total performance overhead, the performance overhead on the execution of both, spatially isolated and non-isolated applications, the resources utilization rate, the waiting time for each application to be mapped, the time spent on the Trusted Manager kernel services including the deployment strategies algorithms, and resource allocation decisions.
- Finally, the comparison of the different deployment strategies according to the performance metric to be optimized and to the considered execution scenario.

The remainder of this article is organized as follows: Section 2 presents some background on cache-based SCAs and current countermeasures. Section 3 presents the targeted architecture and associated threat model, and further explains the spatial isolation countermeasure that we consider in our work. In Section 4, different secure zone deployment and management strategies are explained and discussed. Then, Section 5 presents the baseline strategy as well as the implementation of the proposed deployment strategies. Section 6 presents the evaluation environment and the evaluation setup for the validation of the proposed approaches. In Section 7, results for each proposed strategy are analyzed and compared regarding a set of performance metrics. Section 8 discusses the presented work in terms of security, and previous and future work. Finally, Section 9 draws some conclusions and presents some future work.

2 BACKGROUND

SCAs correspond to an important class of attacks on multitasks systems (Ge et al. 2016). These attacks exploit the leakage of information during or after the execution of the sensitive process from different physical measurements in order to deduce important information about a critical

process (e.g., a cryptographic algorithm) (Blömer and Krummel 2007). When the attacker has no physical access to the platform, analyzing neither the electromagnetic radiation nor the power consumption of the system is possible. However, when physical resources are shared between sensitive and malicious applications, logical SCAs based on the analysis of the execution time of certain operations or memory accesses are still possible. Especially, cache-based SCAs represent a main threat when caches are shared. This section first introduces logical cache-based SCAs. Then, related work is presented.

2.1 Cache-based SCAs

Cache-based attacks may be sophisticated, but their underlying idea is relatively simple: an attacker observes cache-based side-channel information such as the victim's execution time or memory accesses in order to gain information about the sensitive data of the victim process. Additionally, if the attacker can run code on the victim's machine, as well as manipulate the state of the cache, he/she is able to gain extra information. By exploiting this knowledge, the attacker can retrieve confidential data of the critical program (Blömer and Krummel 2007). Among cache-based SCAs, one distinguishes time-driven (or timing) from access-driven attacks.

2.1.1 Time-Driven Attacks. These attacks exploit the vulnerability that, for some algorithms, the execution time is directly related to sensitive data. Moreover, attackers can exploit the fact that the execution time of an application is influenced by the current state of the cache leading to potential leakage of information. There are two categories of time-driven attacks; passive and active. The main difference is the location of the attacker. A passive adversary does not have access to the victim's machine and, thus, cannot manipulate the state of the cache directly. Here, the attacker process triggers the sensitive application (e.g., an encryption algorithm) a certain number of times and measures the execution time. The latter is influenced by the state of the cache, which is itself influenced by each sensitive application execution. These attacks need more samples than active ones and often require statistical methods in order to successfully retrieve the sensitive information. In the work of Bernstein (2005), a passive time-driven attack is remotely performed on the Advanced Encryption Standard (AES) algorithm. On the other hand, an active attacker has access and is able to run code on the victim's machine. This allows him to directly manipulate and probe the state of the cache by filling it with its own data or by evicting some specific cache lines. Here, the attacker can trigger the sensitive application, manipulate the state of the cache, and measure the execution time. This gives to the attacker additional cache information, compared to passive attacks, and leads to more efficient attacks. A well-known technique is the EVICT+TIME (Osvik et al. 2006). Authors perform an active timing attack on AES showing its efficiency compared to the passive attacks presented in Bernstein (2005). Finally, in Bonneau and Mironov (2006), authors improve this type of attack by almost four orders of magnitude compared to the work of Bernstein (2005) by specifically focusing on individual cache collisions during encryption and by attacking the final round of AES encryption instead of the first one in previous work.

2.1.2 Access-Driven Attacks. Access-driven attacks rely on the fact the attacker has access to the victim's machine and that there is a shared cache between the attacker and the victim processes. These attacks exploit the vulnerability that, for some systems, some instructions are related to sensitive data. The principle is to deduce which cache lines the victim has accessed by directly manipulating and probing the shared cache and observing the memory access time. This additional information about the victim's cache accesses makes these attacks more efficient than time-driven ones. PRIME+PROBE is a well-known technique (Percival 2005). Assume that an attacker manipulates the state of the shared cache by accessing some specific memory addresses by filling the cache with its own data (PRIME). Then, the victim runs for a certain time and potentially changes the state of the cache. Finally, the attacker measures the time to access the same memory

addresses again (PROBE). A short access time would indicate that the attacker's data is still in the cache (a cache hit) and, thus, that the victim has not accessed this cache memory line. On the contrary, a large access time would indicate a cache miss which indicates that the victim has accessed the same cache memory line. By exploiting this technique, the attacker infers information about the memory locations accessed by the victim, and thus the instructions and data that have been accessed. These attacks are possible both among the same execution core and across cores.

Among the same execution core: Initially, cache-based attacks were performed through L1 caches in a multithreaded system when two processes, an attacker and a victim one, are concurrently running on the same core and thus share the same L1 cache. In Gullasch et al. (2011), authors implemented this technique against a 128-bit AES implementation of OpenSSL 0.9.8n on Linux.

Across-cores: The focus of cache-based attacks has shifted from first-level to shared LLC (Liu et al. 2015; Irazoqui et al. 2016; Kayaalp et al. 2016), enabling these attacks across cores. The FLUSH+RELOAD technique (Yarom and Falkner 2014) for instance, targets the LLC. Consequently, to launch this attack, the victim and the attacker programs do not need to execute on the same core. This attack extends the technique presented above (Gullasch et al. 2011) with adaptations for multi-core environments. Furthermore, the FLUSH+RELOAD attack is a variant of the PRIME+PROBE technique (Percival 2005) but relies on shared memory pages between the victim and the attacker. Here, the attacker exploits the inclusiveness property of Intel LLC; every data on lower caches is cached as well in the LLC. Consequently, the attacker can evict a specific cache line (e.g., through a specific assembly instruction such as *cflush*) from the LLC which will in turn evict the line from all the lower-level caches. A round of attack of the FLUSH+RELOAD (Yarom and Falkner 2014) technique consists of three phases: In the first phase, a monitored shared memory line is flushed from the cache hierarchy. During the second phase, the attacker waits, letting some time to the victim to execute and to potentially access the monitored cache line. Finally, in the third phase, the attacker reloads the monitored cache line measuring the time necessary to load it. If the victim accessed the cache line during the waiting phase, then the line will be accessible in the cache and the load time measured by the attacker will be short. On the other hand, if the victim did not access the line, when reloaded, the line will be fetched from the main memory and the measured access time will be significantly longer. In Yarom et al. (2014), the authors present some implementations of this technique. However, attackers might request a significant number of reloads, which can be detectable. A variant in order to prevent attackers from reloading, resulting in a faster attack, is to replace the reload phase by a second flush phase (FLUSH+FLUSH) (Gruss et al. 2016). Across-cores access-driven attacks have been proven practical across VMs (Irazoqui et al. 2015). Furthermore, in Reinbrecht et al. (2016b), a PRIME+PROBE-based attack has been implemented on a NoC-based Multiprocessor System-on-Chip (MPSoC). While cache-based SCAs are often performed against cryptographic algorithms, techniques presented above are generic and can be used to eavesdrop other non-cryptographic applications in order to recover sensitive (e.g., personal) information. In Gruss et al. (2016), for instance, authors use the FLUSH+RELOAD technique on eavesdropping keystroke timings.

In this work, we focus on active time-driven and access-driven SCAs. In the next section, existing countermeasures are presented.

2.2 Cache-Based SCAs Countermeasures

We distinguish three different categories according to their main goal.

2.2.1 Modifying the Implementation or Traces of Critical Processes. Application-specific countermeasures have been proposed in order to make sure that the leaked information by the critical applications implementation is independent of secret data. Crane et al. (2015), propose a technique

to transform each program in order to make its trace unique offering probabilistic protection against cache-based SCAs. A countermeasure against access-driven attacks is to modify the implementation of sensitive applications (i.e., cryptographic algorithms) in order to avoid any cache access preventing cache leakage useful for access-based SCAs. Different implementations of some classic cryptographic algorithms have been proposed. In Blömer and Krummel's work (2007), authors focus on the AES algorithm (Daemen and Rijmen 2002) and propose several implementations. Results show that this approach leads to less efficient implementations in terms of execution time. Against time and access-driven SCAs, some works propose to modify the critical applications implementation in order to make them *constant-time*, i.e., which do not branch on secrets and do not perform memory accesses that depend on secrets (Campo 2016). In Barthe et al. (2014), authors give a proof that constant-time programs do not leak confidential information through the cache. However, these solutions are application specific. Alternative approaches are presented below.

2.2.2 Disrupt the Adversary Measurements. A more flexible approach is to allow nonconstant-time implementations but to introduce some disruptions on the possible attacker measurements in order to prevent the extraction of sufficient useful information from caches. One approach is to completely flush all the caches at each context switch (Tromer and Osvik 2010). In this way, cache line monitoring would be ineffective since the attacker will always observe that all lines are evicted from the cache. In Guanciale et al. (2016), this approach is implemented in AES examples on one core execution. However, this approach introduces a significant performance overhead on the execution of all processes (untrusted but also trusted processes) due to the increased cache miss rate (up to one order of magnitude delay overhead). This overhead is added to the cost of the flushing itself.

At the cache-level, in Wang and Lee (2007), authors propose to introduce some uncertainty on the behavior of the cache by randomly permuting cache lines or by randomizing the memory-to-cache mapping. At the NoC level, in the work of Sepulveda et al. (2015), authors introduce random delay on each memory and cache access.

However these solutions offer probabilistic protection only. Other solutions, providing strict isolation in order to avoid any interference between the attacker and victim processes, are presented below.

2.2.3 Avoiding Cache Sharing. Disabling caches seems a naive solution in order to counter cache-based SCAs. Indeed, blocking attackers to use the caches concurrently with sensitive processes would make caches invisible to them and consequently, cache-based attacks would be impossible. However, this countermeasure may entail great performance costs for applications not using the cache.

Another approach is to partition the physical resources, especially the cache. One solution is to exploit a partitioned cache against cache-based SCAs guaranteeing that a sensitive application does not share partitions in cache. This solution avoids cache interference. Partitions are mostly static (Page 2005; Kim et al. 2012). A variant of this approach introducing dynamism on a cloud environment is page coloring (Shi et al. 2011; Raj et al. 2009), which aims at isolating VMs cache dependencies. In this manner, physical memory pages are partitioned among VMs in such a way that no VM shares cache lines with any other VM. Same as page coloring, specific memory lines can be locked in the cache in order to ensure that only a given sensitive application can evict these lines from the cache (Wang and Lee 2007).

Previous approaches have not been originally designed for emergent architectures such as many-core systems. These solutions need to be evaluated and/or revisited in order to be practical and efficient for these new technologies. In contrast, in Méndez Real et al. (2016a), the spatial isolation has been proposed in order to thwart cache-based SCAs on multi- and many-core

systems. This solution takes advantage of the large number of resources available in these systems. The main idea is to temporarily dedicate part of the physical resources (called *secure zone*) to a single sensitive application providing an isolated environment for its execution. This solution avoids any cache sharing with the critical application and ensures the non-interference with any other applications. Consequently, the sensitive application's cache activity cannot longer be monitored and cache-based SCAs can no longer be performed. This countermeasure will be further explained in Section 3.4. In this work, we focus on this last solution and especially on its implementation in order to minimize its performance degradation.

3 CONSIDERED SYSTEM

We consider a many-core accelerator and we focus on the evaluation of different application strategies for the implementation of the spatial isolation solution against cache-based SCAs proposed in Méndez Real et al. (2016a). In this section, the system and associated threat model are explained.

3.1 Many-Core Accelerator

The system considered in this work (Figure 1) is composed of a host machine and a clustered NoC-based many-core accelerator. The host machine runs an OS and executes a great number of applications in parallel. The host machine and the accelerator are connected via a local interface (e.g., a Peripheral Component Interconnect (PCI)). TSAR (TSAR 2014) architecture, Kalray's Massively Parallel Processor Array (MPPA) (Kalray's 2016), Mellanox' TILE-Gx36 (TILE-Gx36 2017), and TILE-Gx72 (TILE-Gx72 2017) processors are some examples of many-core architectures. This work relies on the TSAR architecture (TSAR 2014) used as an accelerator. The latter is composed of clusters interconnected with a 2D-Mesh NoC. Each cluster is composed of four processing elements (PEs), a network interface, and a L2 memory bank (here, the L2 cache is the LLC). While L1 cache is private to each PE, L2 cache is shared among all clusters of the platform. Moreover, the physical address space of the main memory is statically partitioned into a fixed number of segments (equal to the number of clusters in the architecture). Each segment is statically mapped on an L2 memory bank as shown in Figure 1. Consequently, each cluster L2 memory bank is in charge of one memory segment. The memory is then logically shared among all the PEs but physically distributed. Indeed, L2 caches are instantiated close to the cluster's cores. This particularity of the architecture that was initially designed in order to allow a locality-aware deployment is suitable, as well, for the implementation of the spatial isolation. For this work, we consider a monotask PE. This prevents L1 caches sharing. However, processes executing on different processors still share the LLC. When launching an application through the host machine OS, the user specifies whether the application needs to be executed spatially isolated within a secure zone. The host machine is responsible for the launch and execution of applications. However, it can delegate part of the execution workload to the many-core accelerator. Moreover, depending on the user applications requirements, the host machine can request the accelerator manager to create a secure zone on the accelerator. On the accelerator side, a dedicated cluster allows the execution of the manager responsible for the dynamic deployment and execution of applications.

3.2 Case Study Cache-Based Attacks Vulnerability

A study was conducted in order to evaluate the vulnerability to cache-based SCA of applications in a normal execution scenario on a many-core architecture. For this, a TSAR-like (Figure 1) (TSAR 2014) architecture and ALMOS OS (Almaless 2014) are considered. In this scenario, there is no security mechanism implemented. The architecture encompasses a 4×4 2D-mesh NoC connecting 16 clusters: 15 regular clusters encompassing 4 processors, and one manager dedicated cluster encompassing only one processor (61 processors in total). Table 2 summarizes the parameters of the

Table 1. Cache Attacks Vulnerability in Baseline Strategy with 77% Average Resources Utilization Rate

Application (App)	Number of clusters used by this App	Percentage of the App exec. time where clusters have been shared	Applications sharing the cluster with
Application 1	5 clusters	1 cluster shared for 99.4% of the exec. time	Application 2
Application 2	5 clusters	1 cluster shared for 74.6% of the exec. time	Application 1
		1 cluster shared for 99.9% of the exec. time	Application 3
Application 3	5 clusters	1 cluster shared for 83.0% of the exec. time	Application 2
		1 cluster shared for 16.9% of the exec. time	Application 4
		1 cluster shared for 99.9% of the exec. time	Application 5
Application 4	6 clusters	1 cluster shared for 75.6% of the exec. time	Application 3
		1 cluster shared for 37.1% of the exec. time	Application 5
Application 5	8 clusters	1 cluster shared for 15.7% of the exec. time	Application 3
		1 cluster shared for 45.6% of the exec. time	Application 4

architecture used for these experimentations. Five concurrent matrix multiplication applications are considered. Each application executes 256×256 matrix computations. Each application is composed of 17 parallel tasks. A *master* task creates the matrices, splits the computation load, creates 16 *child* tasks, and sends to each the required data. Each child task makes its own computation and sends the results to the master task (see further details in Section 6.2.2). Tasks of each application may be spread across several clusters and thus may use and share several cluster memory banks (one LLC memory bank per cluster) with other applications. Experiment results, summarized in Table 1, present, for each application, the percentage of its execution time where the application shared LLC memory banks, as well as the number of different applications the banks have been shared with. These results highlight the time that each application is vulnerable to cache-based attacks as well as the potential attacker processes (applications sharing LLC memory banks with). It can be concluded that, in this scenario, where the average resources (computing and memory) utilization rate is 77%, resource sharing introduces an important vulnerability for each application (from 15.7% up to 99.9% of their execution time) that needs to be addressed.

3.3 Threat Model

For this work, the physical platform is considered trusted and not physically accessible. Furthermore, we consider that the attacker is able to know the code of sensitive applications running on the accelerator (e.g., cryptographic algorithm for which the code is public) and to trigger its execution, but cannot access it during runtime nor modify it. Finally, this work focuses on cache-based SCAs, which do not require any physical access to the system. The manager on the accelerator executes the minimum services required to control the deployment and execution of the applications (Section 5). It is also responsible for the isolation mechanisms and secure zones deployment strategies. Hence, the software code running on the manager must be trusted (Trusted Manager). This work focuses on the dynamic application deployment, execution, and resource management on the accelerator. Thus, we assume that:

—the manager on the accelerator is trusted (Trusted Manager) and the boot step is protected,
—the communication between the host machine OS and the Trusted Manager is protected,
—the off-chip accesses to external memory and peripherals as well as the NoC communications are secured.

Moreover, in this work, application migration is not considered due to the induced complexity and cost. Indeed, migration would imply the secure remapping of the application and processor

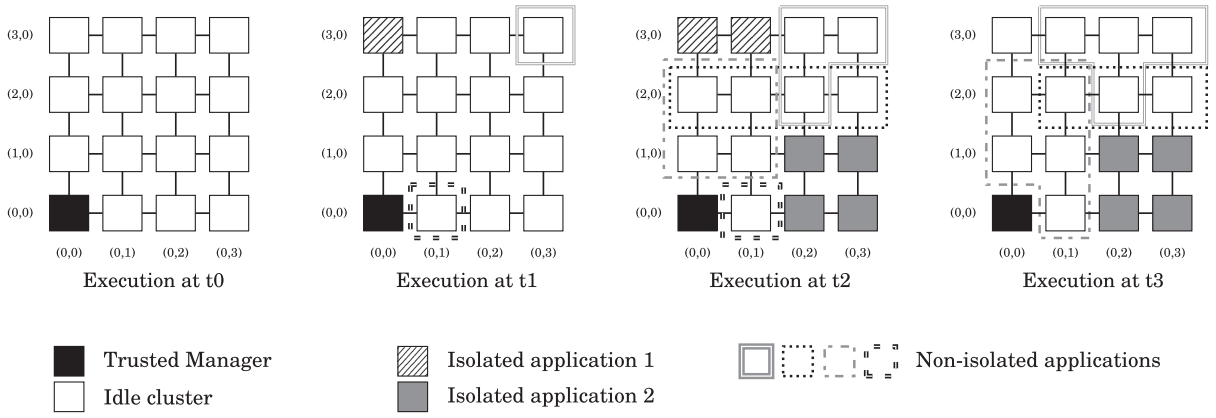


Fig. 2. Principle of spatial isolation.

context switch as well as the memory remapping of the application data and instructions in order to leverage data locality. However, migration might be considered in the future in order to cope with problematics such as dark silicon, component aging, faulty components, and so on. Finally, sensitive and potentially malicious applications may run concurrently on the accelerator.

3.4 Spatial Isolation of Sensitive Applications

In order to thwart cache-based SCAs, we consider the physical isolation of sensitive applications proposed in Méndez Real et al. (2016a). The aim is to prevent cache sharing for every critical application by temporarily dedicating physical resources (*secure zone*) for its execution. A secure zone is composed of a number of clusters that are completely dedicated to a single sensitive application during its entire execution time. Secure zone clusters might be spatially contiguous in order to minimize the NoC latency on the communication between tasks within an isolated application. Secure zones are dynamically deployed, managed, and released. In this way, cache sharing with an isolated application is avoided, and cache-based attacks are no longer possible against isolated applications. Moreover, several isolated applications can run simultaneously (Figure 2), each one within a separate secure zone. Finally, when an isolated application has been executed, memory within its secure zone is cleared in order to avoid any leakage of information. Figure 2 illustrates the principle of this technique for dynamic size secure zones. Different consecutive execution times are shown ($t0 < t1 < t2$). At *t0*, there is no load on the platform. Then, at *t1*, three different applications, including *Isolated application 1* requiring to be spatially isolated, are deployed and start their execution. At that time, there are enough available resources. Consequently, applications do not share resources and do not interfere with each other. Later, at *t2*, previous applications have extended on several clusters, secure zone 1 encompasses now two clusters, which are dedicated and are not shared with any other application. Further, *Isolated application 2* has been deployed and executes within a four clusters secure zone. Other non-isolated applications have been deployed as well. Due to the increasing load on the platform, non-isolated applications are obliged to share their resources with each other. Before deploying a sensitive application, a secure zone is created dedicating a certain number of clusters, depending on the chosen deployment strategy (see Section 4). The sensitive application executes within the secure zone. Every task created by an isolated task is mapped and executed within the secure zone. Depending on the deployment strategy, secure zone resources can be dynamically released. Once the isolated application finishes, its remaining secure zone resources are released and memory within the secure zone is cleared in order to avoid any leakage of information. At this stage, released resources are declared available

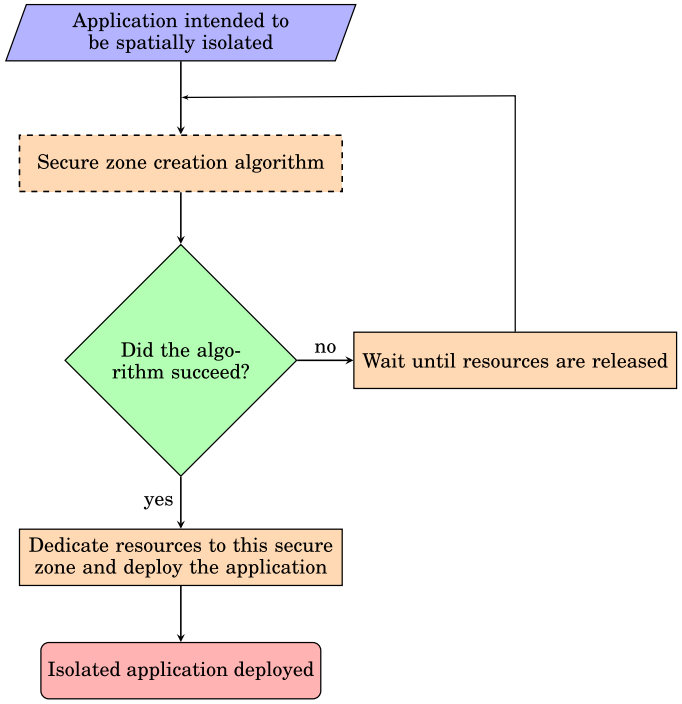


Fig. 3. Creating a secure zone.

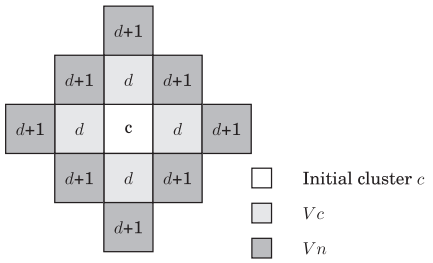


Fig. 4. Exploration of clusters.

again and can be used by other applications. In order to integrate the new services required to dynamically manage the secure zones, the kernel services are extended. Notice that in this approach, the non-isolated applications still use and share caches with other untrusted applications. On the other hand, the dedication of secure zone clusters resources might introduce an under-utilization of resources and, thus, a performance degradation. In this work, we focus on the implementation of this technique in order to minimize and handle the induced performance overhead.

4 DIFFERENT APPROACHES FOR THE DEPLOYMENT OF SECURE ZONES

To implement spatial isolation, the kernel executed by the Trusted Manager requires to be extended with new services responsible for the dynamic management of applications and secure zones.

4.1 Static Approach

A naive approach is to create a secure zone of a static size in terms of clusters. It can be composed of all the needed resources in order to achieve the isolated application maximum parallelism. Or it can be restrained to a limited size. The secure zone size is assumed to be known (e.g., previously determined, specified by the user, or through application profiling). The secure zone is created and all the resources within it are dedicated before the first task of the application can start executing. The secure zone size being fixed, each new task within the application will need to be mapped on the secure zone resources. If the zone includes all the resources the isolated application needs, each time there is a new isolated application task, resources within its secure zone will be available and the task will be mapped without waiting for resources. Once all tasks of the isolated application are finished, all the secure zone resources are released and are declared available again. If there are not enough available resources to create a secure zone, another attempt will be made when a resource is released. Consequently, the application will wait for available resources. Figure 3 shows the flow of the secure zone creation for every deployment strategy, the dashed block corresponds to a secure zone creation algorithm (see Algorithms 1 and 2). Algorithm 1

ALGORITHM 1: Creating a fixed size Secure Zone

```
1:  Input:
2:  l: size of E in terms of number of clusters
3:  A: architecture
4:  Output:
5:  E: list of clusters in the Secure Zone if success,
6:  FAILED: if failure
7:  Let be:
8:  P: list of idle clusters in A
9:  Vc: list of clusters from P to explore at depth d
10: Vn: list of clusters from P to explore at depth d + 1
11: d: current depth
12: c, c': clusters in P
13:
14: while P ≠ [] do
15:     d := 0
16:     Vc := []
17:     E := []
18:     let c being the first cluster in P
19:     Vn := [c]
20:     remove c from P
21:     while Vn ≠ [] do
22:         Vc := sort(Vn), Vn is sorted by the distance from c in ascending order
23:         Vn = []
24:         d = d + 1
25:         while Vc ≠ [] do
26:             let c' be the first cluster of Vc
27:             add c' to E
28:             remove c' from Vc
29:             if size(E) = l then
30:                 return E
31:             end if
32:             for all v a neighbor cluster of c' in P do
33:                 if d<l then
34:                     remove v from P
35:                     add v to Vn
36:                 end if
37:             end for
38:         end while
39:     end while
40: end while
41: return FAILED
```

is responsible for the creation of a fixed-size secure zone. This algorithm is the base for every strategy presented below. Algorithm 1 explains the fixed-size secure zone algorithm with the following notations (lists are spelled in upper-case letters while single elements and variables in lower-case letters):

—A: architecture,
—P: list of idle clusters in A,

— l: required size in terms of number of clusters for the secure zone,
— E: list of secure zone clusters,
— c: an initial cluster from which the secure zone is created (white cluster in Figure 4),
— d: current depth of explored clusters starting from c. Depth d is ranged from 1 to l. Figure 4 shows the cluster exploration space according to d, for l=3. Indeed, when d=1, only the initial cluster c is explored (white cluster in Figure 4). When d=2, the initial cluster c has been explored and its four direct neighbor clusters are explored (light gray clusters in Figure 4). Finally, when d=3, thus, d=l (c and its four direct neighbor clusters have been explored), the neighbor clusters of the neighbor clusters of c are explored (dark gray clusters in Figure 4). In Figure 4, it can be seen that the solution, a zone of l contiguous clusters starting from c, can exist only within the represented zone (when d is less or equal to l, see Algorithm 1, line 33),
— Vc: list of explored clusters from c at current depth d (light gray clusters in Figure 4),
— c': the first cluster in Vc,
— $sort$(list): sort of clusters in a list by the distance between each cluster and the initial cluster c (in ascending order), and
— Vn: list of clusters to explore at depth d+1 (dark gray clusters in Figure 4).

For performance reasons, deployment strategies are based on greedy algorithms aimed at finding a solution, in this case, l idle contiguous clusters, as fast as possible and not necessarily at finding the global best solution. The algorithm receives as input the architecture state as well as the required size of the secure zone (A and l, respectively, in notations above). It gives, as output, a list of l contiguous clusters starting from a given cluster, *initial cluster*, c, in Algorithm 1 if successful (here, E). On the contrary, if the algorithm does not succeed, then the output is a failure notification.

Considering that idle clusters are in list P, the algorithm considers each idle cluster as the initial cluster c (Algorithm 1, line 14), but stops as soon as it finds a solution. Starting from c, it tries to find enough idle contiguous clusters by selecting for each cluster in the secure zone, the idle neighbor cluster, the closest one to the original cluster c. This, in order to minimize the communication cost between the *father task*, that will be mapped on the original cluster c, and the children tasks created by the father task and mapped on the remaining clusters within the secure zone.

First, the algorithm considers the first element in P list (Algorithm line 18), adds it to a list of currently considered clusters Vn (line 19), and removes it from P (line 20) in order to prevent it from considering the same cluster again. Clusters in Vn are sorted in Vc by their Manhattan distance to the original cluster c in an ascending order (line 22). The variable d, indicating the current clusters exploration depth, is increased. The closest idle cluster to c found (first element in Vc), c', is added to the list of secure zone clusters E (line 27) and removed from Vc (line 28). If the required size of E is reached, then the algorithm is finished and E is returned. Otherwise, if the clusters' exploration depth has not reached l, then the idle direct neighbor clusters of c' in P are added to the currently considered clusters in Vn list and removed from P (lines 34, 35). This procedure is repeated until the list Vn is empty. Indeed, an empty Vn list indicates that there is no zone of l idle contiguous clusters starting from c in the architecture A. In this case, a different initial cluster c is selected in P. If every cluster in P has been considered as initial cluster, then there is no secure zone of l idle contiguous clusters. In this case, the algorithm returns a failure notification.

— **Advantages:** In this approach, the size of the secure zone, in terms of clusters, is determined before its deployment. The amount of resources that the secure zone will include will determine the parallelism that the isolated application will be able to achieve. Therefore, the performance of the isolated application can be favored by selecting a suitable secure zone

size. The application will indeed be able to achieve its maximum parallelism if the secure zone is composed of all the resources needed by the isolated application.

— **Drawbacks:** On the other hand, before the isolated application can be deployed, it will wait until the necessary contiguous clusters are available in order to create its secure zone. Thus, we can expect an extra waiting time before the execution of isolated applications that will depend on the load of the accelerator and size of its secure zones. Moreover, since the resources within the secure zone are dedicated to a single sensitive application during the application's entire execution time, an under-utilization of resources within the secure zone is expected. In the same way, the untrusted applications are prevented to use the dedicated resources, which will impact their performance as well as the global performance (i.e., execution time of the set of applications running on the accelerator) due to the potential under-utilization of resources within the secure zones.

4.2 Dynamic Approach

In order to cope with the under-utilization of resources within secure zones, three approaches involving a dynamic secure zone size have been considered.

4.2.1 Fully Dynamic Approach. In this fully dynamic approach, the size of the secure zone is dynamically adapted to the needs of the isolated application according to the load of the platform. For this, the isolated application requires only one single idle cluster to form its initial secure zone (see Algorithm 2), and physical clusters are dynamically added (Algorithm 3) and released from the secure zone. When a new task belonging to an isolated application requires to be mapped, the dynamic approach algorithm first searches for an idle processor within the secure zone. If there is no idle processor within the secure zone, then it searches for an idle cluster contiguous to clusters within the secure zone. If no contiguous cluster is available, then there are two possibilities, either the new task waits until a resource within the isolated application secure zone is released, or the task waits until a resource on the architecture is released and the secure kernel adds it to its secure zone.

— **Advantages:** Isolated applications might wait a shorter time than in a static size approach since an isolated application only needs one single cluster to start executing. Moreover,

ALGORITHM 2: Creating a Dynamic Size Secure Zone

```
1:  Input:
2:  A: architecture
3:  Output:
4:  E: list of clusters in the Secure Zone if success,
5:  FAILED: if failure
6:  Let be:
7:  c: cluster in A
8:
9:  E := []
10: for all c ∈ A do
11:     add c to E if c is idle
12:     return E
13: end for
14: return FAILED
```

ALGORITHM 3: Extending a dynamic size Secure Zone

```
1: Input:
2: E: list of clusters in the Secure Zone,
3: A: architecture
4: c: original cluster on which the father task is mapped
5: Output:
6: E: list of clusters in the Secure Zone if success,
7: FAILED: if failure
8: Let be:
9: P: list of idle clusters in A
10: Vc: list of clusters from P
11: Vn: list of clusters from Vc
12: c': a cluster in E
13: c'' a cluster in Vc
14:
15: for all c' ∈ E do
16:     Vc := []
17:     Vn := []
18:     for all v a non-explored neighbor cluster of c' in P do
19:         add v to Vn
20:     end for
21:     if Vn ≠ [] then
22:         Vc := sort(Vn), Vn is sorted by the distance from c in ascending order
23:         let c'' be the first cluster of Vc
24:         add c'' to E
25:         return E
26:     end if
27: end for
28: return FAILED
```

since the size of a secure zone is dynamically adapted, a better utilization of resources is expected. Consequently, the performance of untrusted applications may be less penalized.

— **Drawbacks:** While this approach might entail better resources utilization and less impact on the untrusted applications performance, isolated applications performance will no longer be a priority. Consequently, we can expect that increasing the secure zones' size will be more difficult.

4.2.2 Hybrid Approach. A variant of the fully dynamic approach explained above, is to dedicate a non-optimized number of clusters (specified by the user, parameter l in Algorithm 1) to an isolated application before executing it, but to dynamically add resources to the secure zone while needed (following the approach presented in Algorithm 3). In this approach, the isolated application needs to wait until the secure kernel finds the minimum number of clusters required to form a secure zone before starting its execution. Once the secure zone is created and the secure kernel launches the execution of the isolated application, clusters can be dynamically added and released, but the minimum specified secure zone size is always guaranteed.

— **Advantages:** This solution guarantees a minimum size of the secure zone, and by consequence, a minimum performance of the isolated application. On the other hand, it also takes into account the current load of the accelerator when trying to dedicate more resources to

the secure zone. The resources utilization rate is thus expected to be better than in a static approach thanks to dynamism. Moreover, untrusted applications are expected to be less penalized than in the static secure zone size scenario since less resources are expected to be dedicated.

— **Drawbacks:** However, this solution may penalize the isolated applications performance since achieving their maximum parallelism is not guaranteed. Finally, this solution requires fixing a minimum secure zone size for each isolated application.

4.2.3 Resource Reservation. An approach in order to favor the dynamic extension of secure zones when there are not enough available resources to create a secure zone of a suitable size (e.g., encompassing all the resources the isolated application needs) is to reserve currently non-available resources in order to prevent other applications from using them once they are declared completely available again. Indeed, when an application intended to be isolated is ready to be mapped and there are not enough available resources to create an optimal size secure zone (referred to as l), the largest available zone is chosen (of size l') and dedicated to the isolated application, which immediately starts its execution. Further, the number of missing clusters on the secure zone (l-l') are selected among contiguous clusters to be *reserved*. These latter need to be contiguous to clusters within the secure zone or, if necessary, to the reserved clusters. Reserved clusters are tagged and when a resource within them is available, it will not be allocated to any other (trusted or untrusted) application. Once all the resources within a reserved cluster are declared available, the kernel adds it to the secure zone if the isolated application still needs it. Moreover, the secure kernel will constantly update the number of required clusters by the isolated application. In the meanwhile, if there are not sufficient clusters within the secure zone, isolated application tasks will need to wait for computing resources in its secure zone to be released. In this case, less reserved clusters would be necessary. Consequently, the secure kernel may dynamically release reserved clusters. As in this work, application migration is not considered (see Section 3.3); once an isolated application starts to be executed, it cannot be migrated when a larger zone than its current secure zone is released. More sophisticated parameters can be used in order to decide which clusters are worthy to reserve. Indeed, the execution time left for processors in each cluster or the number of pending tasks could also be taken into account when selecting clusters to reserve. While this would entail higher and more complex activity on the secure kernel, it would certainly increase the chances of extending the secure zones.

— **Advantages:** In case of a *good* bet, this solution can be very interesting as isolated applications only need one single cluster to start to be executed and reserved clusters allow achieving a good performance of isolated applications. Furthermore, the dynamism of the approach entails *good* resources utilization rates.

— **Drawbacks:** On the other hand, if the bet turns out to be *bad*, this approach can be very penalizing for isolated applications. In fact, if the reserved resources are not released during the execution of the isolated application, then the secure zone will not be extended and new tasks within this application will wait for other tasks within the secure zone to finish in order to start its execution. Consequently, the isolated application may not achieve its maximum performance depending on the load of the accelerator and on the quality of the bet. Moreover, this approach requires a high activity on the trusted kernel compared with a static approach.

5 IMPLEMENTATION: BASELINE STRATEGY AND ITS EXTENSION

The services responsible for the dynamic allocation and management of resources on the accelerator (*kernel services*) are executed on a dedicated processor called *manager*. In their original state,

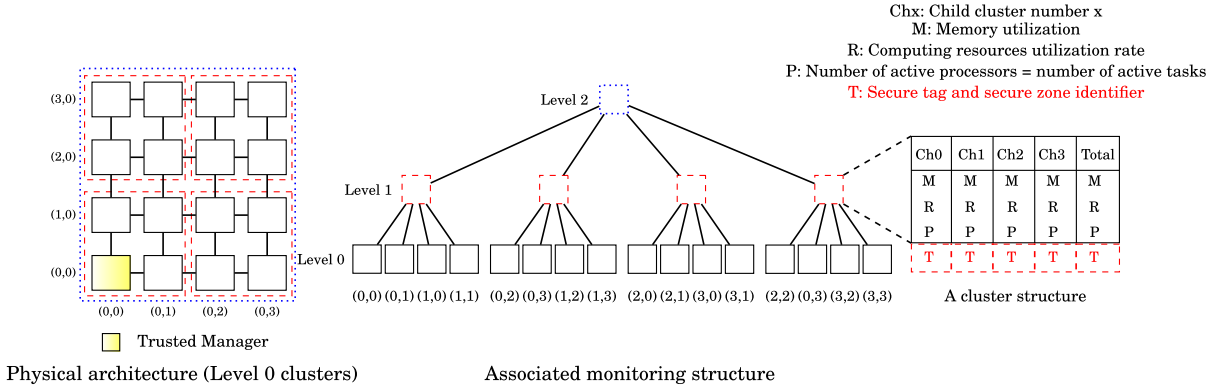


Fig. 5. Overview of the monitoring structure.

they do not encompass any security mechanism and have been designed to favor performance based on the implementation of the ALMOS OS (Almaless 2014) (baseline strategy). These services have been extended in order to integrate the spatial isolation mechanisms proposed in this work. In this section, key kernel services are explained in their original and extended versions.

5.1 Monitoring

The state of the accelerator needs to be constantly monitored in order to dynamically make decisions on the resource allocation. For performance purposes, as in the work of Almaless (2014), a tree structure showing the state of each cluster has been implemented. Figure 5 shows an overview of the monitoring structure for a 16-clusters architecture, each cluster encompassing four processors. In this tree structure, physical clusters (level 0 clusters) are grouped by four, forming a logical cluster (level 1). In the same way, logical clusters are grouped by four, forming an upper level logical cluster (level 2). Each physical or logical cluster is associated with a data structure (*A cluster structure* in Figure 5) containing some parameters describing the state of the resources (the processors and memory utilization rates as well as the number of active processors and tasks). For a physical cluster, the first four columns in its corresponding cluster structure concern the state of the memory and the computing resources within the physical cluster. The fifth column is the sum of the four first columns. On the other hand, for a logical cluster, the first four columns concern the states of the four child clusters (lower level clusters). The fifth column concerns the sum of the first four columns. This organization has been designed for performance purposes as it allows, when visiting the structure from top to bottom, to bypass clusters that do not provide enough resources for the current request. Moreover, it allows a bottom-up approach, as well, in order to locally find available resources from a given physical cluster. While the shape and size of the monitoring structure are static, the parameters within the cluster structures describing the state of the accelerator are regularly updated. Two monitoring updates are implemented. First, a systematic one is made when a task is mapped on a processor or when a processor is released. For this update, it is necessary to visit and update values of the concerned physical cluster(s), but also of each upper level parent (logical) cluster. Second, resources utilization rates are periodically updated. The monitoring structure is consulted each time a decision on resource allocation needs to be made.

Extension: The monitoring structure has been extended in order to associate to each cluster a secure tag indicating if the cluster is dedicated to a secure zone as well as the secure zone identifier (*T* in Figure 5). In the fifth column, this active tag would indicate that all child clusters are already dedicated to a secure zone. Secure tags and secure zone identifiers are taken into account when

taking a resource allocation decision, since these clusters are not available to other applications while tagged secure.

5.2 Mapping New Applications or New Tasks

The new application and new task mapping services are described in this section. First, the algorithm responsible to map new applications aims at finding a *good* processor to map the new task. At most, this algorithm visits the entire monitoring structure. The structure browse starts from the highest level cluster (i.e., the root of the tree structure; level 2 in Figure 5). The aim is to find an idle processor. For this, if according to the fifth column values there is at least one idle processor, the first four columns are visited until one idle processor is found. When it is the case, the child cluster containing the found idle processor is visited. The process is repeated until an idle processor is found on a physical cluster. If there is no idle processor on the entire platform, the task is added to the pending tasks list and it will wait for a processor to be released. Second, a task may ask for the mapping of *child* tasks. Parent and child tasks are expected to communicate together. In order to favor performance, the mapping algorithm aims at minimizing the communication costs. For this purpose, a child task must be mapped the closest possible to its parent task. Indeed, the task mapping algorithm searches for an idle processor starting from the physical cluster of the parent task (level 0 cluster). If no idle processor is found in this cluster, the algorithm goes up and searches on the logical cluster containing the parent physical cluster (level 1 cluster). The algorithm consults, then, the fifth column of the logical cluster structure, and if there is at least one idle processor in one of the child clusters (level 0), it goes down to the found cluster. On the other hand, if there is no idle cluster, the algorithm goes up to the parent cluster (level 2 logical cluster). This process is repeated until either one idle processor is found, or the entire structure has been visited and no idle processor is found. In this last case, the task is added to the pending tasks list and it will wait for a processor to be released.

Extension: These two services have been extended considering that when deploying a new application or task, the Trusted Manager knows whether it is intended to be isolated. If the new application or task to be mapped does not require to be isolated, then the *original* mapping algorithm is used. However, resources within clusters tagged secure in the monitoring structure are considered temporarily not available for non-isolated applications. Consequently, the original service algorithm exploration zone might be reduced since clusters tagged secure are not visited. On the other hand, if the application or task requires to be isolated, then the Trusted Manager uses one of the new algorithms for the creation and management of a secure zone proposed in this work according to the considered strategy presented in Section 4.

5.3 Releasing Secure Clusters

When an isolated application is finished, all the clusters remaining within its secure zone are released. Additionally, in dynamic secure zone size approaches, clusters are dynamically released. When a cluster from a secure zone is released, different steps are taken. First, memory within the cluster as well as remanent information (buffers, registers, state of the caches) need to be cleared in order to avoid any leakage of information. Second, the cluster is declared available again. This latter step requires to modify the secure tag T in the monitoring structure (Figure 5), for the corresponding cluster structure. Finally, the monitoring structure is updated in order to propagate the new information.

The extension of the kernel services including the different strategies presented in Section 4 have been implemented and compared via a virtual prototyping environment.

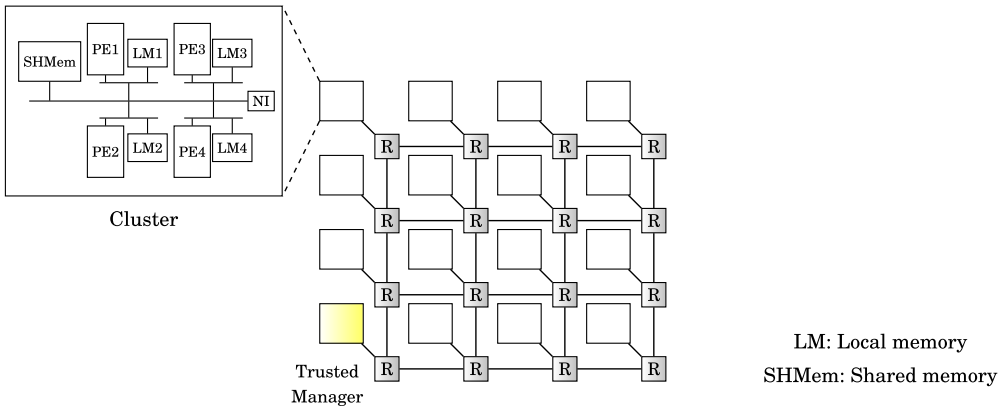


Fig. 6. Overview of the prototyped system.

6 EVALUATION ENVIRONMENT AND EXPERIMENTAL SETUP

In this section, the virtual prototyping tool used for these evaluations and the experimental setup are presented.

6.1 Evaluation Environment

The extension of the kernel manager services, including the proposed deployment strategies, have been evaluated and compared through the MPSoCSim tool (Wehner et al. 2015). MPSoCSim is based on the Open Virtual Platforms (OVP) technology (OVP 2017), coupled with SystemC models (SystemC 2017). It allows the evaluation of clustered NoC-based multi- and many-core architectures (Méndez Real et al. 2016b). MPSoCSim relies on a system level modeling SystemC NoC where each router is connected to a SystemC Transaction Level Modeling standard (TLM-2.0) Network Interface (NI) connected to a local group referred to as a *cluster*. Each cluster can encompass up to four OVP processor models, each one connected to its local memory and shared resources within the cluster such as shared memory and an NI.

6.2 Experimental Setup

6.2.1 MPSoCSim Setup. For the evaluation of the experimentations presented in this work, a 2D mesh 4×4 NoC-based system has been prototyped (Figure 6). The NoC is composed of 15 *regular clusters* of four processors each, and one cluster encompassing one single processor dedicated to the execution of the kernel manager services. The system encompasses 60 processors and 1 processor dedicated to the kernel manager services. Each processor is connected to a private and shared memory that emulate L1 and L2 memory banks, respectively. Table 2 summarizes the system parameters used for the experimentations. Notice that the chosen NoC parameters are identical to those used for the validation of the MPSoCSim tool (Wehner et al. 2015), providing reliable results.

6.2.2 Execution Scenario. In this section, the Trusted Manager, the evaluated applications, and execution scenarios are presented.

— Many-core accelerator Trusted Manager:
For these experimentations, one processor is dedicated and acts as the Trusted Manager of the accelerator. It computes the decision algorithms for the dynamic deployment of the applications (see Section 5), as well as the services needed for creating and handling secure zones (Section 4).

Table 2. System Parameters Used for the Evaluations

Processor parameters	**Chosen value**
Cortex A9 ARM Frequency	667MHz
MicroBlaze (MB) Frequency	100MHz
Nominal MIPS	100
Real flit time (ARM)	850ns
Real flit time (MB)	40ns
System memory parameters	**Chosen value**
L1 cache size	64KBytes
LLC Size	3MBytes
NoC parameters	**Chosen value**
Network frequency	100MHz
2D Mesh network size	4×4 clusters
Number of processors per regular cluster	4
Number of processors in total	60MBs + 1 ARM
Router clock delay pass through	1 cycle

—Applications:
We consider that, when launching an application, the user specifies whether it requires to be isolated within a secure zone. Synthetic applications with a high degree of parallelism such as matrix multiplications are used. Each application corresponds to 256×256 matrix computations. A first task (*master* task), generates the matrices, splits the computation, and dynamically sends the data to each mapped *slave task*. Slave tasks can be mapped on the cluster executing the master task or on distant clusters according to the load of the platform. Slave tasks access to the memory within their cluster, perform the corresponding computation, and send back to the master task the generated results. When a task is finished, its PE is released and the Trusted Manager flushes the corresponding memory partition. A 17 parallel tasks application allows the study of an unfavorable scenario for the considered system. Indeed, when the size of the secure zone is fixed to achieve the application maximum parallelism, or when there are sufficient available resources, the secure zone will be composed of 5 clusters out of 16 in this considered case. However, only 17 processors out of 20 will be used. In consequence, three out of four processors on the fifth cluster will not be used, which represents an unfavorable case for the considered architecture. Applications are duplicated in order to increase the workload on the accelerator. All the applications are supposed to be ready to execute from the beginning of the execution scenario. However, a priority level is associated to each application (master task) to determine in which order the ready tasks are served. For these experimentations, 5 applications, each one composed of 17 parallel tasks, are concurrently executing. This is 85 concurrent tasks in total and an average resources utilization rate of 77% when there is no isolated application.

—Evaluated secure zone deployment and management strategies:
The deployment strategies explained in Section 4 have been evaluated and compared:
  —**baseline strategy** This scenario corresponds to the minimum services presented in Section 5. It does not include any security mechanism and no application is isolated.
  —**Strategy A.1.** Secure zones with a fixed size encompassing all the resources needed by the spatially isolated application in order to achieve its maximum parallelism. In the case considered in this work, the application parallelism is 17 tasks running in parallel and

clusters are composed of four PEs. Consequently, the secure zone size in this case encompasses five clusters (20 dedicated PEs in total).
- —**Strategy A.2.** Secure zones with a fixed restrained size limited to four clusters.
- —**Strategy B.1.** Fully dynamic approach.
- —**Strategy B.2.** Hybrid approach. In this case, we fixed the guaranteed minimum secure zone size to two clusters.
- —**Strategy B.3.** Resource reservation.

—Evaluated execution scenarios:
For the evaluation of the performance overhead induced by each proposed strategy, different execution scenarios have been considered.
- —**Single zone, priority 1 scenario.** One single application (out of five) requires to be spatially isolated within a secure zone. In this first scenario, this application has the highest priority (1 out of 5, 1 being the highest and 5 the lowest priority). Consequently, when mapped, there is no load on the platform. This is the best scenario for an isolated application since it will be mapped the first one, then when there is no load on the accelerator.
- —**Single zone priority 4 scenario.** One single application needs to be isolated. However, unlike the previous scenario, this application has a medium priority (4 out of 5). This allows to take into account the load of the accelerator when trying to create the secure zone.
- —**Multiple zones, multiple priorities scenario.** In this last scenario, the load of the accelerator, as well as the number of applications requiring to be isolated, are considered. In fact, 3 out of 5 applications, with priority levels of 1, 3, and 5, respectively, require to be spatially isolated.

7 RESULTS

In this section, results of each pair of deployment strategies and each execution scenario are presented. Then, results are summarized and strategies are compared according to several performance metrics.

7.1 Results Organization

Applications introduced in Section 6.2.2 are first run concurrently without any secure-enable mechanism (*baseline strategy*). Then, applications are concurrently run according to each pair of secure zone deployment strategy and execution scenario. Finally, since approaches are deterministic, experiments are run once. The main objective is to compare the different deployment strategies for the implementation of the spatial isolation in terms of induced performance overhead and to analyze them according to different performance metrics:

- —total execution time for the set of applications,
- —average execution time of spatially isolated applications,
- —average execution time of non-isolated applications,
- —average computing resources utilization rate,
- —execution time spent on the kernel services impacted by the spatial isolation mechanisms,
- —average time the isolated applications wait before being mapped.

Apart from resources utilization rates in Table 3, overhead results are presented in terms of percentage compared to the baseline strategy. Results in Figures 7 to 11 and Table 3 allow the comparison of each performance metric for each pair of secure zone deployment strategies and execution scenario (Section 7.2). Then, Figures 12 to 14 gather results of every performance metric classified by the execution scenario.

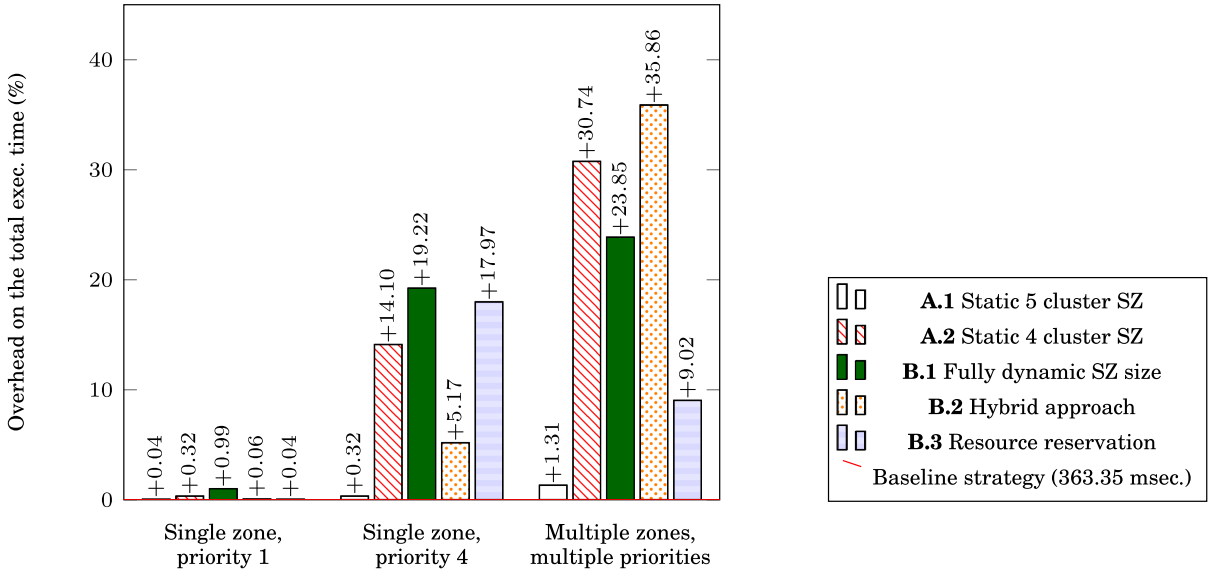


Fig. 7. Total execution time overhead (as a percent) compared to the baseline strategy.

7.2 Comparison According to Each Performance Metric

First, notice that the execution scenario **Single zone, priority 1** is particular. In fact, in this scenario, the application intended to be spatially isolated is served as the first one since it has the highest priority. Consequently, when the manager deploys it, there is no load on the accelerator. As a result, strategies with a guaranteed minimum secure zone size (**B.2.**) and with resource reservation (**B.3**) give similar results than five cluster static secure zone size strategy **A.1**. Indeed, in **B.2** strategy, only the kernel services execution time and total performance overhead are slightly different. Regarding **B.3**, it turns out to be identical than **A.1** since the algorithm finds a secure zone encompassing all the resources needed by the application (five clusters secure zone) directly without requiring to reserve any cluster. In this section, results for each performance metric are presented.

The **total execution time** overhead for each pair of secure zone deployment strategies, and the execution scenario, is presented in Figure 7. While the dynamism in the considered scenarios makes it difficult to explain every aspect of the results, several observations can be made. First, according to these results, the static five clusters Secure Zone (SZ) (**A.1.** strategy) turns out to be the best solution in the evaluated scenarios providing the lowest overhead on the total execution time, almost negligible when there is no load on the accelerator (0.04%). While limiting the size of the SZ to four clusters (**A.2.**) entails a higher overhead than a five cluster SZ, the rest of the strategies do not seem to follow any trend but depend on the execution scenario. In order to better understand and compare these results, it is important to take into account that the total execution time is mostly impacted by both the applications' (isolated and non-isolated) execution time as well as the time spent on the Trusted Manager services for the execution of the deployment mechanisms. Finally, notice that applications and the Trusted Manager run in parallel. In order to compare the impact of different deployment strategies, **the execution time of both isolated and non-isolated applications** are highlighted in Figures 8 and 9, respectively. First, in Figure 8, it can be seen that in scenario **Single zone, priority 1**, results are negative. As these results represent an overhead, negative ones indicate a better performance (reduced execution time) for isolated application(s) compared to the baseline strategy when the considered application was not isolated. This is explained by the exclusiveness of cluster resources and by the fact that, since the isolated application has the highest priority in this scenario, it is deployed as the first one when

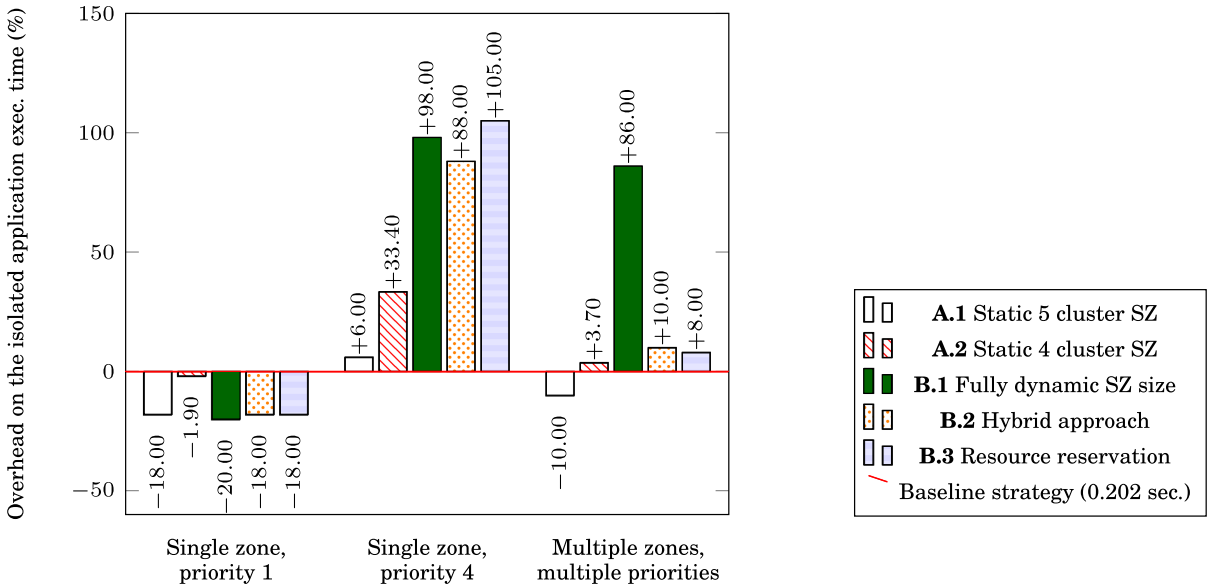


Fig. 8. Overhead on the average execution time of **isolated** applications, as a percent, compared to the average application execution time in the baseline strategy.

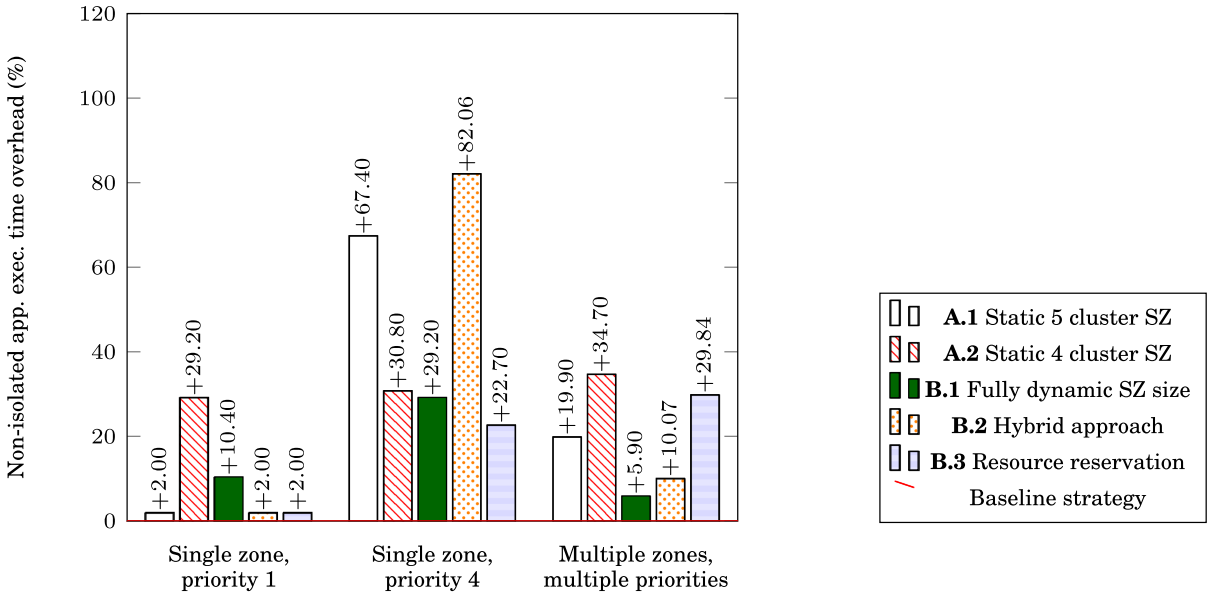


Fig. 9. Overhead on the average execution time of **non-isolated** applications, as a percent, compared to the average application execution time in the baseline strategy.

there is no load on the platform. Indeed, contrary to the baseline strategy, isolated applications do not share cluster resources, which avoids resource concurrency. Moreover, due to the highest priority of the isolated application, all the strategies, except for **A.2.**, achieve the optimal size for the secure zone (five clusters in this case), statically (i.e., **A.1.** strategy), or dynamically (**B** strategies). Consequently, the isolated application achieves its maximum parallelism, which entails achieving better performance. In the **A.2.** strategy, on the contrary, the size of the secure zone is fixed to a non-optimal size (four clusters in this case), regardless of the fact that at the deployment time, all the resources are available on the accelerator. Furthermore, the static five cluster SZ strategy (**A.1.**) always achieves a very good performance of isolated applications but penalizes non-isolated

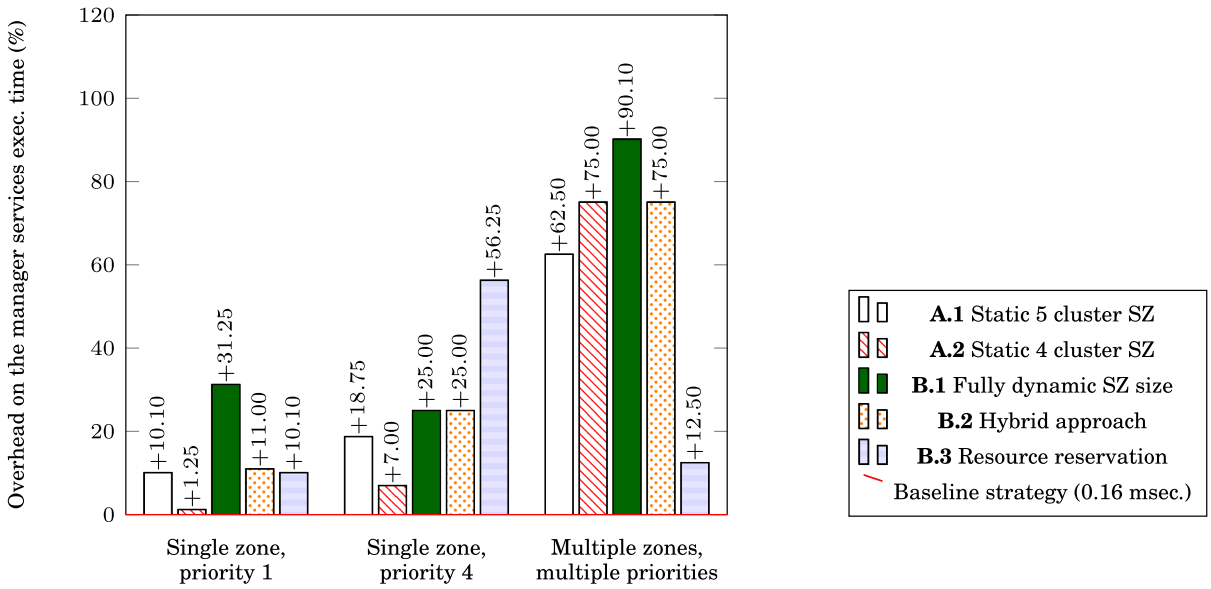


Fig. 10. Time spent on the Trusted Manager kernel services impacted by the secure-enable mechanisms, as a percent compared to the baseline strategy.

applications. Moreover, when limiting the resources within the secure zone (**A.2.**), the performance of isolated applications is lower than in the first strategy for every scenario. However, limiting the secure zone size to four clusters (**A.2.**) instead of five further penalizes non-isolated applications execution time. This is because less resources are dedicated (only four clusters) and thus not available for non-isolated applications, but for a longer time. Indeed, in this case, isolated tasks need to wait for available resources within the secure zone, entailing longer execution time and thus longer resources dedication time. Furthermore, the fully dynamic strategy (**B.1**) and hybrid approach (**B.2.**) tend to penalize isolated over non-isolated applications. However, the hybrid approach guarantees a minimum number of resources for isolated applications so they can achieve better performance.

On the other hand, the **time spent by the Trusted Manager** kernel on the services impacted by the isolation mechanisms proposed in this work are presented in Figure 10. These services are: mapping of each application and task (both isolated and non-isolated), dynamic allocating and releasing resources, as well as creating, managing, and releasing secure zones. Results are presented in terms of induced overhead in percentage of the time spent on the baseline strategy. Dynamic strategies require a higher activity on the Trusted Manager compared to static secure zone size strategies. Moreover, resource reservation results greatly vary according to each scenario. This is because its performance totally depends on the quality of the bet when selecting the resources to reserve. Smarter selection metrics would improve results and would make them more steady (see Section 7.3). Furthermore, according to the secure zone deployment strategy and to the load of the accelerator, isolated applications require to **wait before they can be deployed**. In fact, in a static secure zone scenario, the Trusted Manager will wait until there are enough available resources before it can create a secure zone. Figure 11 shows the average time that isolated applications wait before being deployed for each deployment strategy and execution scenario. It can be noticed that strategies able to guarantee a number of dedicated resources to the secure zone introduce an extra delay depending on the accelerator load. Finally, although overheads in Figure 11 seem significantly penalizing compared to the baseline scenario, these are negligible compared to the application execution time (up to 2.5%) and are already included in the total execution time

Table 3. Computing Resources Utilization Rate within SZ as well as in Total (*Total*)
The resources utilization rate in the baseline strategy is 77%.

	Single zone, pr. 1		Single zone, pr. 4		Mult. zones, Mult. priorities	
	SZ	**Total**	**SZ**	**Total**	**SZ**	**Total**
Static approach, optimal size (5 clusters)	85%	68.5%	85%	71%	85%	61.6%
Static approach, limited size (4 clusters)	65%	64%	65%	69%	65%	55%
Fully dynamic	85%	72%	89%	69%	92%	67%
Hybrid approach	85%	68.5%	81%	60%	85%	55.6%
Resource reservation	85%	68.5%	85.4%	67%	86.2%	65%

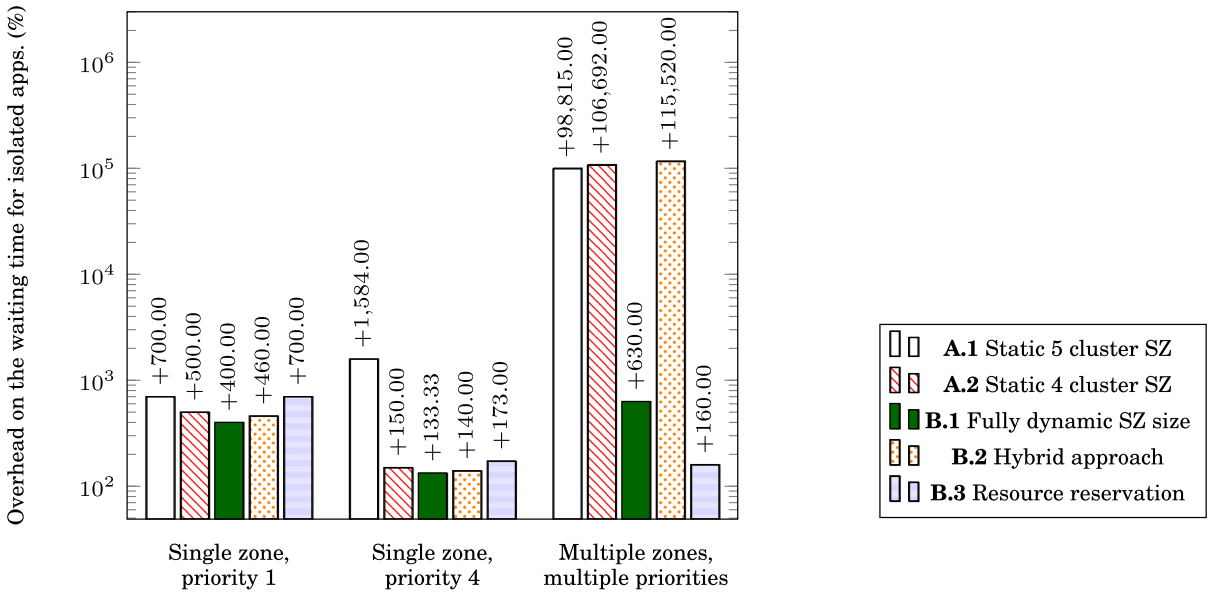


Fig. 11. Overhead on the average waiting time before the deployment of **isolated** applications, as a percent, compared to the baseline strategy.

for the set of applications running on the accelerator in Figure 7 as well as in the execution time of the isolated applications in Figure 8.

In Table 3, the **computing resources utilization rate** is compared in total as well as within dedicated clusters to secure zones for each deployment strategy and execution scenario. While the resources utilization rate within secure zones allows the comparison between different deployment strategies with each other, the resources utilization rate in total shows the overhead of each deployment strategy compared to the baseline strategy (with a resources utilization of 77%). In this table, for each column, the best and worst rate are highlighted in light and dark gray, respectively. It can be noticed that since the fully dynamic strategy (**B.1.** strategy) adapts the resources the best to the needs of applications and load of the accelerator, it achieves the best resources utilization rates.

7.3 Strategies Comparison According to Each Execution Scenario

Figures 12, 13, and 14, gather the results presented above in order to allow the comparison of the strategies according to each different performance metric for each execution scenario. The values

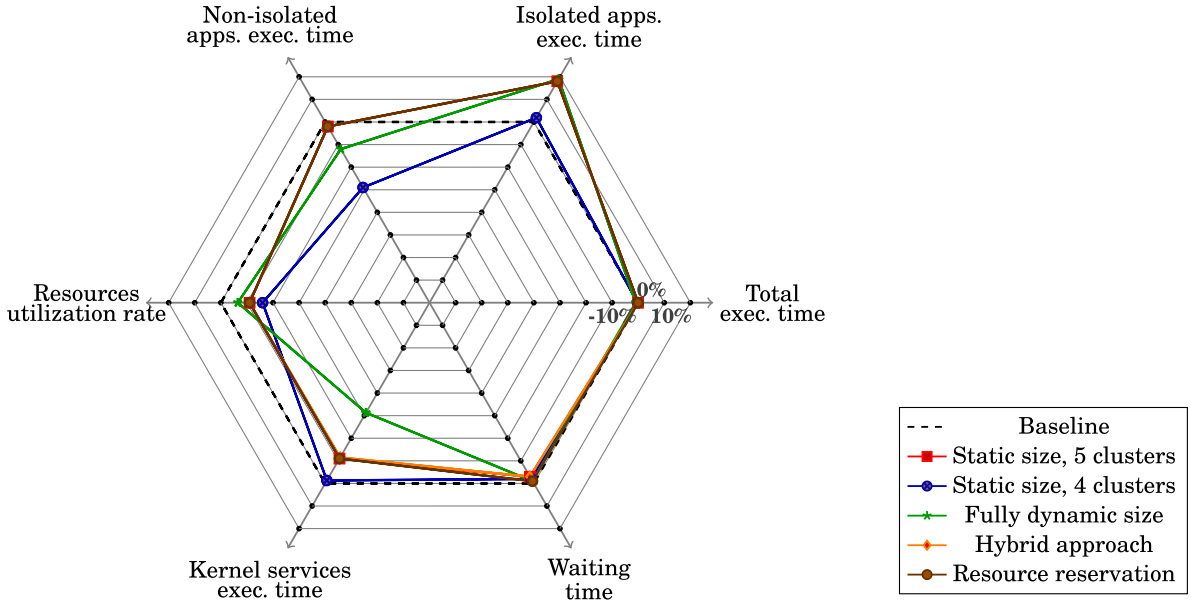


Fig. 12. Single zone, priority 1 scenario. Values presented in terms of induced overhead (as a percent) compared to the baseline value. **Arranged scale**: the closest to the chart, the better. Scale: 1 division : 10%

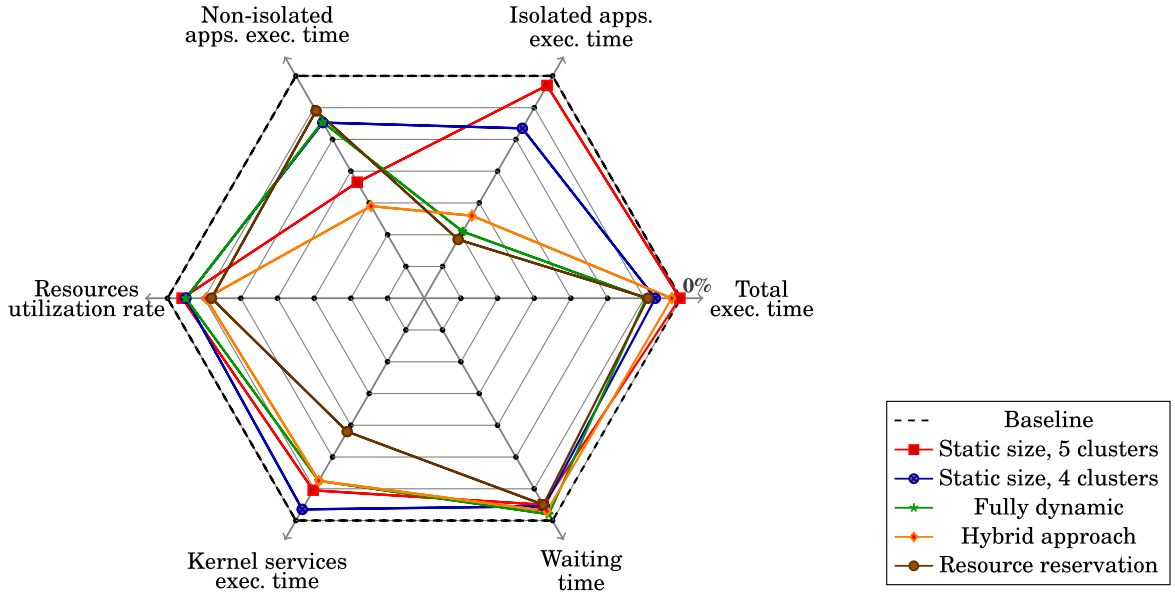


Fig. 13. Single zone, priority 4 scenario. Values presented in terms of induced overhead (as a percent) compared to the baseline value. **Arranged scale**: the closest to the chart, the better. Scale: 1 division : 20%.

of each performance metric of Section 7.1 are presented in terms of induced overhead in percentage compared to the baseline strategy (dashed line at 0% on the chart). Moreover, the scale is arranged in order to present results in such a way that, for each parameter (each axis), the closer a value is to the chart border, the better. Finally, for readability reasons, results concerning the *waiting time* are presented in their *log* value.

—Static approach: When the size of the secure zone is well chosen, this approach is the best solution for the performance of isolated applications. However, when the secure zone size is not well chosen, for example in the **A.2.** strategy, applications performance (both isolated

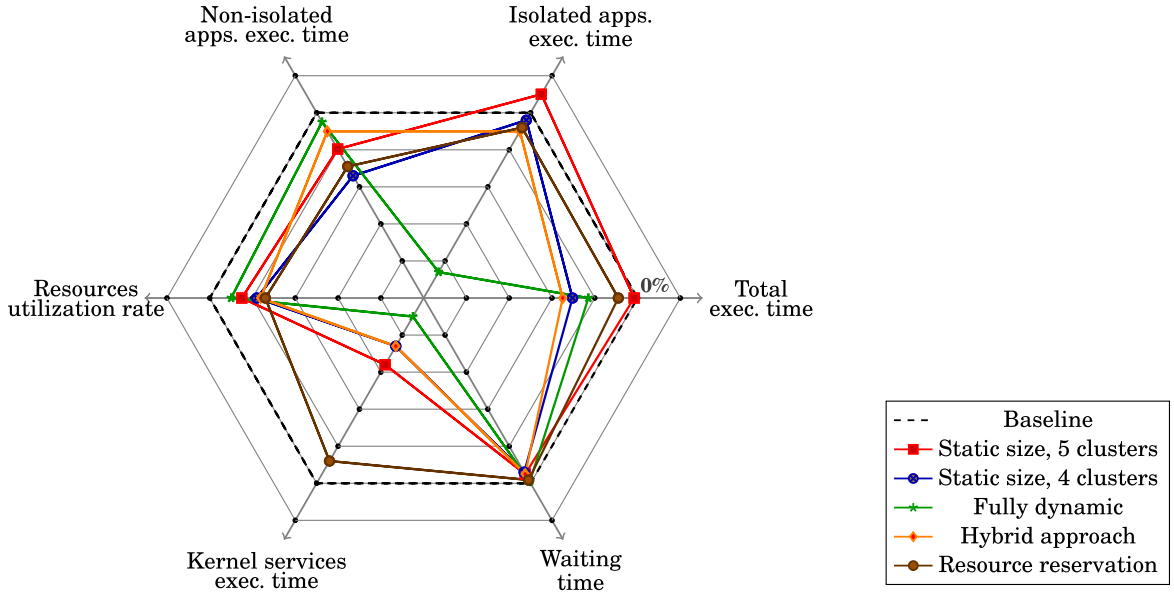


Fig. 14. Multiple zones, multiple priorities scenario. Values presented in terms of induced overhead in percent compared to the baseline value. **Arranged scale**: the closest to the chart, the better. Scale: 1 division : 20%.

and non-isolated) may be well penalized. In fact, in **A.2.** strategy, only four clusters instead of five are dedicated to the sensitive application. Unlike what could be expected, this solution further penalizes more non-isolated applications than in **A.1.** because the resources are dedicated much longer since some isolated application tasks require to wait for other tasks within the same secure zone to release their resources. Consequently, the isolated application execution time is much longer. On the other hand, in static secure zone size approaches, isolated applications require to wait longer to start their execution depending on the availability of resources. However, once they are mapped, they may achieve a very good performance. In conclusion, this approach is suitable when the performance of isolated applications is a priority; however, it requires a very good knowledge of the isolated applications in order to choose a good size of the secure zone.

—Fully dynamic approach: This approach is the best one when it is required to maximize the resources utilization rates within the secure zones and in total, and when the performance of isolated applications is not a priority. Indeed, it does not entail a significant performance overhead on the total execution time (total execution time overhead from 0.99% in scenario **Single zone, priority 1**, up to 23.85% in scenario **Multiple zones, multiple priorities**). However, it tends to penalize the performance of isolated applications and entails a high activity on the Trusted Manager.

—Hybrid approach: This approach is a good tradeoff between fully dynamic and completely static approaches. It guarantees a minimum number of dedicated resources to the isolated application, thus a minimum performance. Also, this approach takes into account the load of the accelerator when trying to extend the secure zones. As a result, non-isolated applications are less penalized than in **A.1.** strategy. Moreover, due to its dynamic side, this approach achieves a good resources utilization rate, but entails more activity on the Trusted Manager compared to a static approach.

—Resource reservation: This approach is interesting but requires more sophisticated metrics when selecting resources to reserve. Indeed, the size of the secure zone as well as the performance of the isolated application depend on the load of the accelerator when creating a

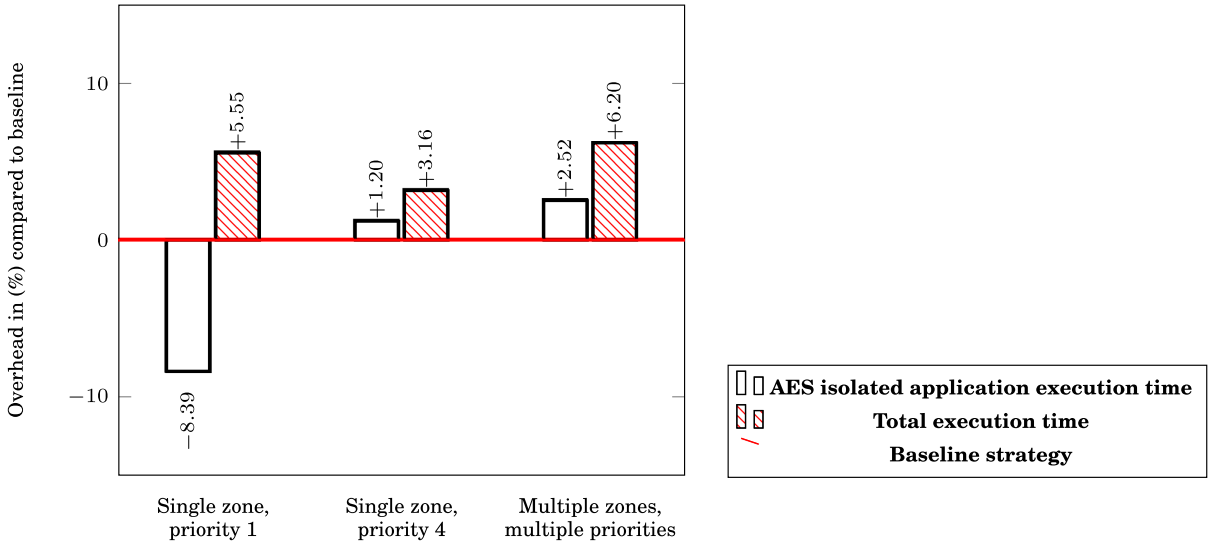


Fig. 15. Overhead of the spatial isolation mechanisms in % compared to the baseline strategy.

secure zone, and on the bet. In this approach, no performance is guaranteed. More complex reservation metrics are necessary in order to increase the chances of extending a secure zone and achieving good performance of isolated applications. However, it would lead to higher complexity of the kernel algorithms.

7.4 Further Experimentations with Different Applications

In this section, two different applications were considered in order to study the impact of the spatial isolation mechanisms on a second execution scenario. For these experimentations, an AES encryption and decryption based on axTLS implementation (axTLS 2016) (AES implementation 2017) is used in addition to matrix multiplications. As for previous experiments, five concurrent applications with different priorities are considered. First, one task-application is performing an AES encryption and decryption of a 64-bytes message following the Cypher Block Chaining (CBC) mode with a 256-bits key (test vectors were taken from the following reference: (test vectors 2001)). Additionally, matrix multiplications are used (see description in Section 6.2.2). Three different scenarios are evaluated; in every scenario, only the AES application is isolated. In **Single zone, priority 1** and **Single zone, priority 4** scenarios, an AES application, additionally to four matrix multiplications are executed. Moreover, the AES application is isolated on a one cluster-secure zone and has priorities first and fourth, respectively. Finally, in the third scenario (**Multiple zones, multiple priorities**) three AES and two matrix multiplications are executed. AES applications have priorities first, third, and fifth, respectively. Finally, each scenario was compared to their **baseline** scenario in which no application is isolated. Figure 15 shows the performance overhead of the isolation mechanisms in scenarios **Single zone, priority 1** and **Single zone, priority 4** on both the total execution time for performing the set of considered applications, as well as on the execution of the isolated application. Results are presented in terms of percentage compared to the baseline scenario. These results show little overhead on the total execution time, up to 5.55% in Single zone, priority 1 scenario, that increases with the number of secure zones (up to 6.2% in **multiple zones, multiple priorities**). However, similarly to results in Section 7.2, the performance of the isolated application is less impacted (up to 1.2% in **Single zone, priority 4** scenario) and can even be improved according to the load on the platform due to the dedication of resources to its execution (down to −8.39% in **Single zone, priority 1** scenario).

8 DISCUSSION

This section first discusses results regarding the threat model (see Section 3.3). Then, further attacks not considered in this work are discussed. Finally, the positioning of this work regarding our previous work and some possible leads for future work are presented.

8.1 Back to the Threat Model

Physical resources isolation is achieved through secure zones. Unlike results presented in Table 1, each sensitive application executes within its secure zone preventing any cluster resource sharing with other applications. In this way, caches used by an isolated application are no longer shared (0% of caches sharing). Consequently, considered cache-based SCAs in this work are no longer possible since secure zone cache activity can no longer be monitored by potential attacker processes. Notice that this solution requires the kernel to be trusted since the latter is responsible for the handling of secure zones. Finally, inter-application communication and off-chip accesses are considered trusted, as well.

8.2 Further Attacks

8.2.1 Spatial Isolation Properties. In this work, we study the spatial isolation on many-core architectures against cache-based SCAs. However, the proposed spatial isolation introduces further security properties. Indeed, besides cache-based SCAs, the spatial isolation of applications running in secure zones prevents, as well, Denial of Services (DoS) attacks on the resources within the secure zones. In fact, the Trusted Manager, is responsible for dedicating some resources to a secure zone for one single sensitive application and to allocate non-dedicated resources only to other applications. Thus, the Trusted Manager guarantees that attacker processes are no longer able to use the dedicated cluster resources, and, as a consequence, DoS attacks on these resources are no longer possible. However, other resources such as the NoC are not DoS free and further mechanisms, in order to protect these resources, are necessary. On the other hand, isolated applications themselves could perform DoS attacks by requesting a huge number of resources to be dedicated as well as by not releasing the dedicated resources. In this work, we have not considered any mechanism preventing the isolated applications from performing DoS attacks. Monitoring the activity of isolated applications and/or forcing them to release the resources after a given time, for example, could be considered in order to detect and/or prevent these kind of attacks.

8.2.2 Further Confidentiality and Integrity Attacks. Spatial isolation counters cache-based SCAs in which the access to data is indirect by monitoring the cache activity and inferring sensitive information. However, attacks through illegal direct access to the memory (reading or writing) are still possible. State-of-the-art countermeasures such as Memory Management Unit (MMU) and/or Memory Protected Unit (MPU) are compatible with our work.

8.2.3 NoC Attacks. According to the threat model (Section 3.3), the communication through the NoC is considered secured. In fact, despite resource dedication within secure zones clusters, the NoC resources are still shared between all the applications. Some literature works consider the problem of information leaked through NoC communication (Wang and Edward 2014; Reinbrecht et al. 2016a; Sepulveda et al. 2015). However, little work can be found on practical attack implementations. In Reinbrecht et al. (2016b), authors propose a timing attack on NoC on a shared LLC MPSoC platform. However, this attack relies on a single LLC shared between all the processors on the platform, which is not the case in many-core for scalability reasons (e.g., memory bottleneck). Indeed, in the system considered in this work, the LLC is distributed among the clusters, which makes it very difficult to analyze NoC traffic. To our best knowledge, there is no attack proving their practicality on distributed LLCs many-core architectures. However, some literature

countermeasures addressing NoC attacks are compatible with our work. For instance, using semi-adaptive routing such as west-first logic instead of the traditional deterministic XY logic, proposed in Sepulveda et al. (2015), in order to disturb the attacker, observations on the NoC traffic can be used in the context of our work. This solution has been implemented and used together with the spatial isolation mechanisms (for the deployment strategy **A.1** and execution scenario Single zone, priority 1), showing an overhead on the total execution time of 2%. Another solution in order to isolate the NoC communication within secure zones would be to use adaptive routing in order to avoid any non-isolated application from using the NoC resources within secure zones. These techniques have been used mostly as fault-tolerance mechanisms in NoC-based architectures. In the work of Fuguet Tolero (2016), for instance, reconfiguration-based recovery mechanisms have been studied on the TSAR architecture (TSAR 2014) in order to allow the system to work in spite of faulty NoC elements such as routers and links. A similar approach could be used in our context, in order to isolate secure zones NoC communications by seeing dedicated resources as temporary faulty components for non-isolated applications.

8.3 Contributions Compared to Our Previous Work

In this work, we have focused on the implementation of the spatial isolation proposed in Méndez Real et al. (2016a). We have extended the secure zone deployment and management strategies initially proposed with new strategies offering a tradeoff between completely static and dynamic approaches. These new strategies offer flexibility compared to previous solutions, and they are able to achieve better performance on the total execution than a completely dynamic approach (resource reservation strategy induces around 1.5% and 15% less performance overhead than fully dynamic strategy in execution scenarios **Single zone, priority 4** and **Multiple zones, multiple priorities**). Also, they are able to achieve less penalized isolated applications performance on execution scenario **Multiple zones, multiple priorities** (resource reservation strategy induces 78% less overhead on the isolated applications performance than fully dynamic approach). Finally, each strategy has been further studied according to the considered execution scenario and to the performance metric to optimize.

8.4 Future Work

In this work, we have considered contiguous secure zones in order to favor isolated application NoC communications. As a consequence, this tends to favor the isolated application performance, which in turn tends to minimize the time that secure zone's resources are dedicated, and thus, the time that non-isolated applications are prevented from using them. Moreover, this approach would make easier the isolation of the secure zone application NoC communication if required (see Section 8.2.3). However, considering non-contiguous secure zones would give more flexibility when creating and extending secure zones. On the other hand, isolated applications being spread onto distant clusters, their communication cost would increase, which will, in turn, impact their execution time and the resources dedication time. The interest of this approach compared to contiguous secure zones would be interesting to study in terms of induced overhead on both isolated and non-isolated applications and is certainly worthy to consider. Moreover, considering migration of secure zones and/or non-isolated applications would as well give more flexibility to our approach since it would be possible to dynamically rearrange allocated resources both dedicated to a secure zone and non-dedicated ones. Finally, the protection of off-chip accesses to memory and peripherals is another possible lead for future work.

9 CONCLUSION

This work focuses on the spatial isolation of sensitive applications on a many-core accelerator in order to thwart cache-based SCAs proposed in Méndez Real et al. (2016a). Particularly, we focus

on its implementation in order to provide a flexible solution minimizing the induced performance overhead. A dedicated processor runs the Trusted Manager of the accelerator and is responsible for the dynamic deployment of applications and secure zones. Sensitive applications are executed within a secure zone spatially isolated from other applications preventing any resource sharing within the dedicated clusters. As a consequence, their caches activity can no longer be monitored and cache-based SCAs can no longer be performed against them. The Trusted Manager kernel services have been enhanced in order to integrate the proposed mechanisms. Several new secure zone deployment strategies have been proposed in order to minimize the induced performance overhead. The proposed strategies have been evaluated according to several performance metrics. Results show that these new strategies offer a tradeoff between completely static and dynamic secure zone size approaches. Non-contiguous secure zones, secure zones migration, as well as the protection of off-chip accesses are some possible leads worthy to consider in future work.

REFERENCES

Ghassan Almaless. 2014. *Operating System Design and Implementation for Single-Chip cc-NUMA Many-Core*. Ph.D. Dissertation. Université Pierre Marie Currie (PMC), France.

axTLS embedded SSL. 2016. Retrieved from http://axtls.sourceforge.net/.

AES implementation in ~300 lines of code | C Code Blog. 2017. Retrieved from https://ccodeblog.wordpress.com/2012/05/25/aes-implementation-in-300-lines-of-code/.

Gilles Barthe, Gustavo Betarte, Juan D. Campo, Carlos Luna, and David Pichardie. 2014. System-level non-interference for constant-time cryptography. In *Proc. of the Conference on Computer and Communications Security*. ACM, 1267–1279.

Daniel J. Bernstein. 2005. *Cache-Timing Attacks on AES*. Technical Report. Retrieved from https://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

Johannes Blömer and Volker Krummel. 2007. Analysis of countermeasures against access driven cache attacks on AES. *Selected Areas in Cryptography* 4876 (2007), 96–109.

Joseph Bonneau and Ilya Mironov. 2006. Cache-collision timing attacks against AES. In *Proc. of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 201–215.

Paolo Burgio, Marko Bertogna, Ignacio Sanudo Olmedo, Paolo Gai, Andrea Marongiu, and Michal Sojka. 2016. A software stack for next-generation automotive systems on many-core heterogeneous platforms. In *Proc. of the Euromicro Conference on Digital System Design (DSD)*. IEEE.

Juan Campo. 2016. *Formally Verified Countermeasures Against Cache Based Attacks in Virtualization Platforms*. Ph.D. Dissertation. Montevideo: UR.FI.INCO.

Stephen Crane, Andrei Homescu, Stephan Brunthaler, Per Larsen, and Michael Franz. 2015. Thwarting cache side-channel attacks through dynamic software diversity. In *Proc. of the Annual Network and Distributed System Security Symposium, (NDSS)*. IEEE, 142–151.

Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael: AES-The Advanced Encryption Standard* (1st ed.). Springer-Verlag, Berlin.

Dmitry Evtyushkin, Jesse Elwell, Meltem Ozsoy, Dmitry Ponomarev, Nael Abu-Ghazaleh, and Ryan Riley. 2016. Flexible hardware-managed isolated execution: Architecture, software support and applications. *IEEE Transactions on Dependable and Secure Computing (TDSC)* PP (2016), 1.

César Fuguet Tolero. 2016. *Introduction of Fault-Tolerance Mechanisms for Permanent Failures in Coherent Shared-Memory Many-Core Architectures*. Ph.D. Dissertation. Université Pierre Marie Currie (PMC), France.

Quian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2016. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering (Cryptogr Eng)* 1–27 (2016), 1.

Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. FLUSH+FLUSH: A fast and stealthy cache attack. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assesment (DIMVA)*. Springer.

Roberto Guanciale, Hamed Nemati, Christoph. Baumann, and Mads Dam. 2016. Cache storage channels: Alias-driven attacks and verified countermeasures. In *Proc. of the Symposium on Security and Privacy (SP)*. IEEE.

David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache games-bringing access-based cache attacks on AES to practice. In *Proc. of the Symposium on Security and Privacy (SP)*. IEEE, 490–595.

Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. S$A: A shared cache attack that works across cores and defies VM sandboxing and its application to AES. In *Proc. of the Symposium on Security and Privacy (SP)*. IEEE.

Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cross processor cache attacks. In *Proc. of the 11th Asia Conference on Computer and Communications Security (ASIA CCS)*. ACM, 353–364.

Kalray's. 2016. MPPA. Retrieved from http://www.kalrayinc.com/kalray/products/.

Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. 2016. A high-resolution side-channel attack on last-level cache. In *Proc. of the 53rd Annual Design Automation Conference (DAC)*. ACM, 72.

Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. 2012. STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud. In *Proc. of the 21st Security Symposium*, USENIX (Ed.).

Fangfei Liu, Yuval Yarom, Quian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *Proc. of the Symposium on Security and Privacy (SP)*. IEEE, 605–622.

Maria Méndez Real, Philipp Wehner, Vincent Migliore, Vianney Lapotre, Diana Goehringer, and Guy Gogniat. 2016a. Dynamic spatially isolated secure zones for noc-based many-core accelerators. In *Proc. of the International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, IEEE (Ed.).

Maria Méndez Real, Philipp Wehner, Jens Rettkowski, Vincent Migliore, Vianney Lapotre, Diana Goehringer, and Guy Gogniat. 2016b. MPSoCSim extension: An OVP simulator for the evaluation of cluster-based multicore and many-core architectures. In *Proc. of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE.

Dag A. Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: The case of AES. In *Proc. of the RSA Conference Cryptographers Track (CT-RSA)*.

OVP. 2017. Open Virtual Platforms. Retrieved from https://www.ovpworld.org/.

Dan Page. 2005. *Partitioned Cache Architecture as a Side-Channel Defense Mechanism*. Cryptology ePrint Archive, Report 280.

Colin Percival. 2005. Cache missing for fun and profit. In *BSDCan 2005*.

Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. 2009. Resource management for isolation enhanced cloud services. In *Proc. of the 2009 ACM Workshop on Cloud Computing Security (CCSW)*. ACM, 77–84.

Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, and Johana Sepulveda. 2016a. Gossip noc – avoiding timing side-channel attacks through traffic management. In *Proc. of the Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE.

Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, Georg Sigl, and Johanna Sepulveda. 2016b. Side channel attack on NoC-based MPSoCs are practical: NoC prime+probe attack. In *Proc. of the 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE.

Martha J. Sepulveda, Jean-Philippe Diguet, Marius Strum, and Guy Gogniat. 2015. NoC-based protection for SoC time-driven attacks. *IEEE Embedded Systems Letters* 7, 1 (2015), 7–10.

Jicheng Shi, Xiang Song, Haibo Chen, and Binyu Zang. 2011. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *Proc. of the 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 194–199.

OVP & SystemC. 2017. Open Virtual Platforms Imperas Software Limited. Retrieved from http://www.ovpworld.org/technology_systemc.

Nist AES test vectors. 2001. Recommendation for block cipher modes of operation: methods and techniques-nistspecial-publication800-38.a.pdf. Retrieved from http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf.

TILE-Gx36. 2017. Mellanox Products: TILE-Gx36 Processor. Retrieved from http://www.mellanox.com/page/products_dyn?product_family=237&mtag=tile_gx36.

TILE-Gx72. 2017. Mellanox Technologies -End-to-End Connectivity for HPC and Data Center Server and Storage. Retrieved from http://www.mellanox.com/page/products_dyn?product_family=238&mtag=tile_gx72.

Eran Tromer and Dag A. Osvik. 2010. Analysis of countermeasures against access driven cache attacks on AES. *Journal of Cryptology* 23, 1 (2010), 37–71.

Pham Trung-Dung, Nguyen Van-Tien, and Nguyen Truong-Son. 2016. Development of a many-core architecture for automotive embedded systems. *Journal of Automation and Control Engineering* 4, 2 (2016), 147–152.

TSAR. 2014. Retrieved from https://www-soc.lip6.fr/trac/tsar.

TSUNAMY. 2016. The TSUNAMY project. Retrieved from https://www.tsunamy.fr.

Yao Wang and Suh G. Edward. 2014. Cache games-bringing access-based cache attacks on AES to practice. In *Proc. of the 6th International Symposium on Networks on Chip (NoCS)*. IEEE/ACM.

Zhenghong Wang and Ruby B. Lee. 2007. New cache designs for thwarting software cache-based side channel attacks. In *Proc. of the Symposium on Computer Architecture (ISCA)*. IEEE, 494–505.

Philipp Wehner, Jens Rettowski, and Diana Goehringer. 2015. MPSoCSim: An extended OVP simulator for modeling and evaluation of network-on-chip based heterogeneous MPSoCs. In *Proc. of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE.

Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proc. of the Security Symposium*, USENIX (Ed.). 719–732.

Received April 2017; revised September 2017; accepted November 2017

10 Détection logicielle d'attaques par canaux auxiliaires exploitant les mémoires caches

Les travaux présentés dans la section précédente se sont appuyés sur la notion d'isolation en réservant un ensemble de ressources dédiées à l'exécution d'applications sensibles. Cependant, cette approche n'est pas toujours réalisable. Par exemple, certaines ressources, comme les mémoires caches de dernier niveau des architectures de processeur Intel sont partagées par tous les coeurs du processeur. C'est dans ce contexte, qu'entre 2016 et 2019, Le travail de thèse de Mme Maria Mushtaq s'est focalisé sur la détection logicielle d'attaques par canaux auxiliaires exploitant les mémoires caches au sein des architectures de processeur Intel.

10.1 Approche proposée

L'approche proposée vise à détecter un comportement malveillant trahissant une attaque en s'appuyant sur des algorithmes d'apprentissage. Pour cela, le matériel et le logiciel collaborent afin de réaliser une supervision de métriques issues des compteurs de performances du processeur (HPC). En cas de détection, le système d'exploitation pourra interrompre le processus à l'origine de l'attaque.

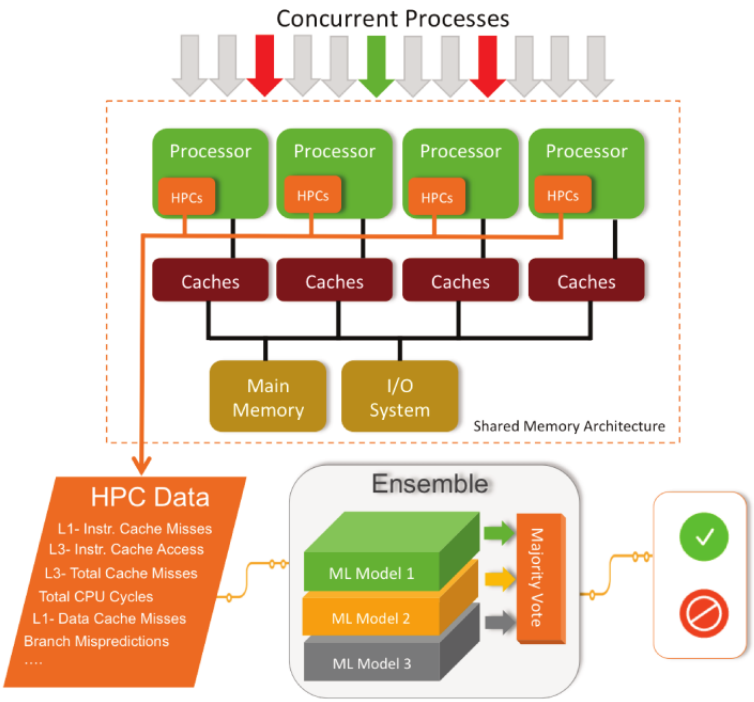


FIGURE 17 – Vue d'ensemble de l'approche proposée pour la détection logicielle d'attaques par canaux auxiliaires exploitant les mémoires caches

La figure 17 présente une vue d'ensemble de l'approche proposée. Dans un premier temps, des informations provenant des compteurs de performances concernant principalement l'utilisation des mémoires caches et la performance sont agrégées et présentées à différents modèles préalablement entraînés à détecter les signatures des attaques par canaux auxiliaires exploitant les mémoires caches. Dans un second temps, ces modèles transmettent une décision individuelle afin de procéder à un vote majoritaire. L'issue de ce vote est le résultat final du mécanisme de détection.

Dans le but de sélectionner les compteurs de performances les plus pertinents pour détecter un ensemble de techniques d'attaques exploitant les mémoires caches, des expérimentations

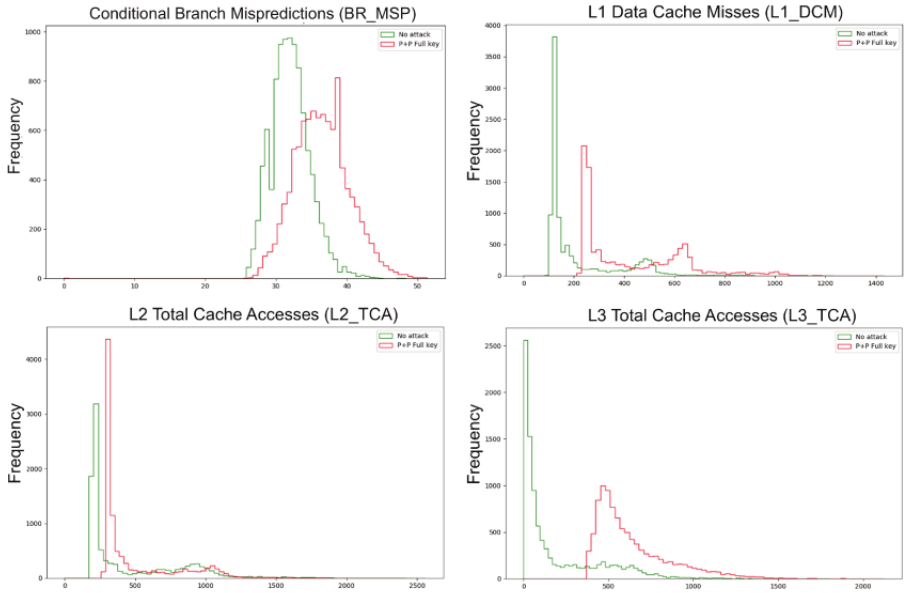


FIGURE 18 – Résultats d'expérimentations l'impact sur les HPC d'une attaque de type PRIME+PROBE lors d'un chiffrement AES

permettant de quantifier l'impact des attaques sur les métriques disponibles via les compteurs de performances ont été réalisées. La figure 18 illustre le résultat de ces expérimentations pour quatre métriques lors de l'exécution d'un chiffrement AES sans attaque (courbe verte) et lors de l'exécution d'une attaque de type *PRIME+PROBE* (courbe rouge) dans des conditions idéales pour l'attaquant (c.-à-d. sans charge supplémentaire s'exécutant sur la plateforme). On remarque que pour chaque métrique considérée (le nombre d'erreur de prédiction des branchements conditionnels, le nombre de *miss* dans le cache de données L1 et les nombres d'accès aux caches L2 et L3), l'effet de l'attaque est visible.

Ces expérimentations ont permis d'identifier quatre métriques pouvant être utilisées pour la détection des attaques de type *FLUSH+RELOAD*, *FLUSH+FLUSH* et *PRIME+PROBE* : Le nombre total de cycles CPU, le nombre de *miss* du cache de données L1, le nombre d'accès au cache L3 et le nombre de *miss* du cache de données L3. Le mécanisme de détection proposé s'appuie sur ces métriques pour détecter en temps réel une éventuelle attaque. Pour cela 13 modèles de machine learning (linéaires et non-linéaires) ont été entraînés pour détecter les trois techniques d'attaques considérées dans ces travaux. La liste des modèles considérés est la suivante : *Linear Regression* (LR), *Linear Discriminant Analysis* (LDA), *Support Vector Machine* (SVM), *Quadratic Discriminant Analysis* (QDA), *Random Forest* (RF), *K-means*, *K-Nearest Neighbors* (KNN), *Nearest Centroid*, *Naive Bayes*, *Perceptron*, *Decision Tree* (DT), *Dummy Non-linear* et *Neural Networks*.

Enfin, dans le but de renforcer la qualité du mécanisme de détection, les modèles RF, DT et SVM sont également utilisés conjointement dans un module nommé *ensemble*. Dans ce module, les trois décisions fournies par les modèles pré-entraînés sont propagées à un module de vote majoritaire qui détermine la décision finale.

10.2 Principaux résultats

La figure 19 illustre la capacité de détection de ces différents modèles pour détecter ces attaques lors de l'exécution d'un chiffrement AES dans différentes conditions de charge : NL : charge nulle (seuls les services de l'OS s'exécutent en parallèle des programmes victime et attaquant), AL : charge moyenne (2 applications supplémentaires issues de la suite SPEC s'exécutent en parallèle des programmes victime et attaquant), FL : charge importante (4 applications supplé-

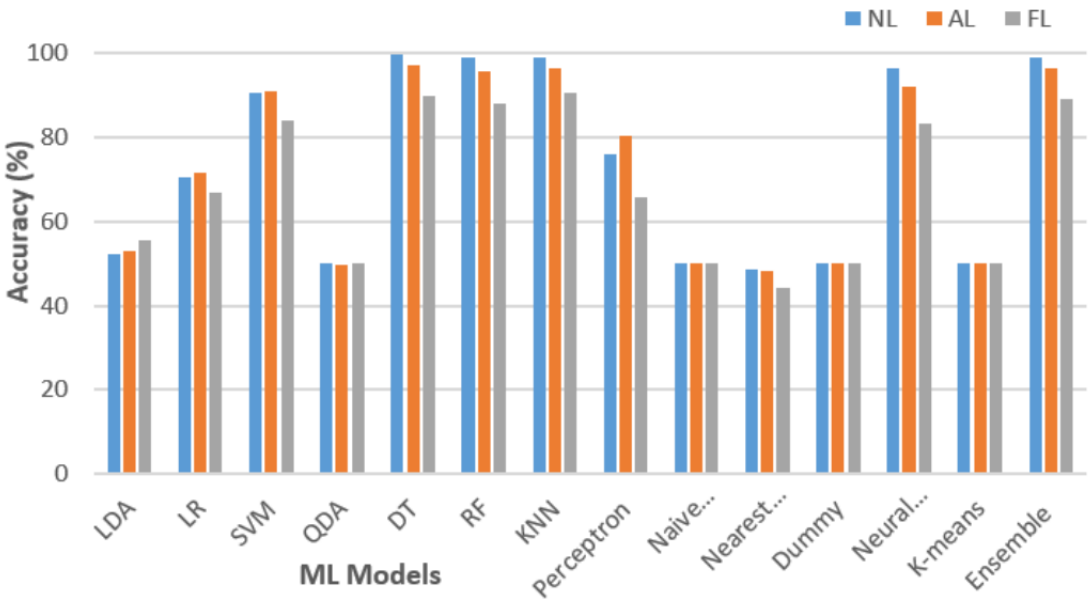


FIGURE 19 – Comparaison des modèles de machine learning considérés en terme de précision

mentaires issues de la suite SPEC s'exécutent en parallèle des programmes victime et attaquant). On observe que les modèles *SVM DT*, *RF*, *KNN* et *neural network* proposent une précision de détection importante. Malheureusement, bien que performant, une solution de détection en temps réelles basée *KNN* et *neural network* serait trop coûteuse.

TABLE 12 – Résultats pour l'utilisation des modèles *SVM DT*, *RF* individuellement et au sein du modèle *Ensemble* pour la détection des attaques *PRIME+PROBE* ciblant le chiffrement AES

Modèles	Charge	précision (%)	temps de détection	FP (%)	FN (%)	Surcoût (%)
SVM	NL	99,99	0,21	0,01	0,00	7,83
	AL	99,82	0,21	0,18	0,00	
	FL	94,92	0,21	5,08	0,00	
DT	NL	99,99	0,21	0,01	0,00	6,59
	AL	99,76	0,21	0,24	0,00	
	FL	95,00	0,21	5,00	0,00	
RF	NL	99,99	0,21	0,01	0,00	11,3
	AL	99,72	0,21	0,28	0,00	
	FL	92,67	0,21	7,33	0,00	
Ensemble	NL	99,99	0,21	0,01	0,00	8,03
	AL	99,77	0,21	0,23	0,00	
	FL	97,62	0,21	2,37	0,01	

Dans le but d'évaluer l'intérêt du module *Ensemble* proposé dans nos travaux, il est nécessaire de réaliser une analyse plus fine de la performance des modèles pour la détection des attaques. La table 12 présente les performances détaillées pour le cas de la détection des attaques de type *PRIME+PROBE* ciblant le chiffrement AES. Concernant la précision de la détection, on remarque que le module *Ensemble* permet une amélioration des performances dans le cas où les conditions de charge sur la plateforme sont importantes. La précision est alors supérieure à 97% contre 94,92 %, 95% et 92,72% pour les modèles *SVM DT*, *RF* utilisés seuls. Ce gain est obtenu grâce à la diminution du nombre de faux positifs qui est ramené à 2,37% quand l'utilisation

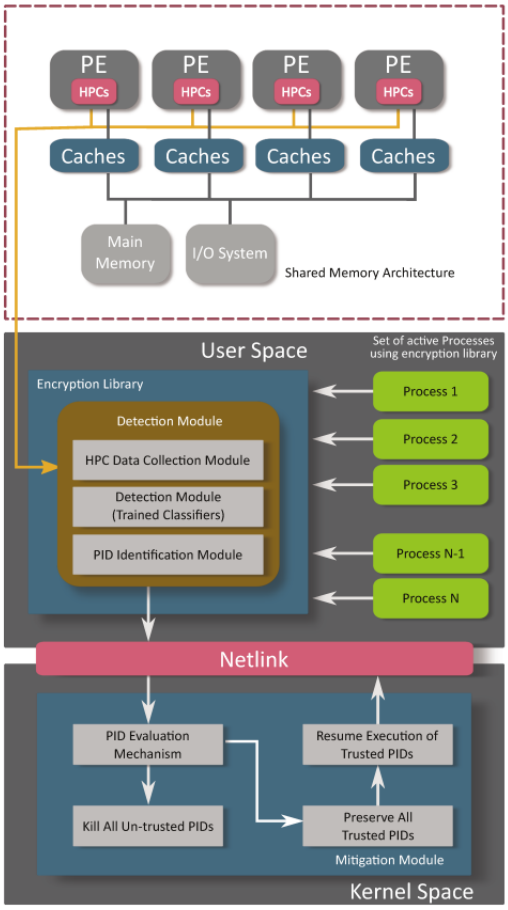


FIGURE 20 – Vue d'ensemble de l'approche intégrant, au sein du système d'exploitation Linux, un mécanisme de réaction face à des attaques par canaux auxiliaires exploitant les mémoires caches

d'un modèle seul engendre un taux de faux positifs supérieur à 5%. L'influence du module sur le nombre de faux négatifs est négligeable dans ce cas de figure. De plus, on remarque que le temps de détection ne change pas quel que soit le modèle. Il correspond à 0,21% du temps nécessaire à la réussite de l'attaque. Enfin, le surcoût sur les performances globales engendré par la surveillance en temps réel du système est comparable quel que soit le modèle utilisé.

L'ensemble des résultats produits dans le cadre de cette étude est disponible dans l'article [Mushtaq 2020a] publié dans la revue IEEE Access en 2020 et disponible à la suite de cette section du document.

10.3 Intégration au sein d'un système d'exploitation Linux

Le mécanisme de détection reposant sur l'acquisition de métriques issues des HPC du processeur et sur l'utilisation d'un ensemble de modèles de machine learning présenté dans les sections précédentes a été intégré au sein d'un mécanisme de sécurité plus global pour le système d'exploitation Linux. Ce nouveau mécanisme fonctionne en deux étapes distinctes. Dans un premier temps, le module de détection, embarqué au sein d'une brique logicielle sensible (e.g. une librairie cryptographique), est utilisé pour alerter le système en cas de détection d'un comportement malveillant. Dans le cas où une alerte est levée, cette dernière est transmise à un module de réaction s'exécutant dans l'espace noyau. C'est le point de départ de la seconde étape. Les identifiants des processus (PID) utilisant la brique logicielle surveillée sont recherchés. Ces processus sont alors suspendus par le système d'exploitation. Une phase d'identification des PID est alors réalisée

pour trier les processus de confiance (généralement les processus système) des processus potentiellement malveillants (généralement les processus utilisateurs). Le système met alors un terme aux processus potentiellement malveillants puis relance les processus de confiance. La figure 20 illustre l'approche proposée. Le lecteur intéressé pourra se référer à l'article [Mushtaq 2022] publié dans la revue Annals of Telecommunications en 2022 pour une description détaillée de l'approche.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

WHISPER
A Tool for Run-time Detection of Side-Channel Attacks

MARIA MUSHTAQ[1], JEREMY BRICQ[2], MUHAMMAD KHURRAM BHATTI[3], AYAZ AKRAM[4], VIANNEY LAPOTRE[5], GUY GOGNIAT[5], AND PASCAL BENOIT[1]

[1]LIRMM, Univ Montpellier, CNRS, Montpellier, France
[2]University of Brussels, Brussels, Belgium
[3]ECLab, Information Technology University, Lahore, Pakistan
[4]University of California Davis, CA, USA
[5]Lab-STICC, Université Bretagne Sud, Lorient, France

Corresponding author: Maria Mushtaq (e-mail: maria.mushtaq@lirmm.fr).

This work is partially supported by the NCCS and PHC PERIDOT Project e-health.SECURE (Grant ID: 3-6/HEC/R&D/PERIDOT/2017).

ABSTRACT

High resolution and stealthy attacks and their variants such as Flush+Reload, Flush+Flush, Prime+Probe, Spectre and Meltdown have completely exposed the vulnerabilities in Intel's computing architecture over the past few years. Mitigation techniques against such attacks are not very effective for two reasons: 1) Most mitigation techniques protect against a specific vulnerability and do not take a system-wide approach, and 2) they either completely remove or greatly reduce the performance benefits of resource sharing. In this work, we argue in favor of detection-based protection, which would help apply mitigation only after successful detection of the attack at runtime. As such, detection would serve as the first line of defense against such attacks. However, for a detection based protection strategy to be effective, detection needs to be highly accurate, to incur minimum system overhead at runtime, should cover a large set of attacks and be capable of early stage detection, *i.e.*, at the very least before the attack is completed. We propose a machine learning based side-channel attack (SCA) detection tool, called WHISPER that satisfies the above mentioned design constraints. WHISPER uses multiple machine learning models in an Ensemble fashion to detect SCAs at runtime using behavioral data of concurrent processes, that are collected through hardware performance counters (HPCs). Through extensive experiments with different variants of state-of-the-art attacks, we demonstrate that the proposed tool is capable of detecting a large set of known attacks that target both computational and storage parts in computing systems. We present experimental evaluation of WHISPER against Flush+Reload, Flush+Flush, Prime+Probe, Spectre and Meltdown attacks. The results are provided under variable system load conditions and stringent evaluation metrics comprising detection accuracy, speed, system-wide performance overhead and distribution of error (*i.e.*, False Positives & False Negatives). Our experiments show that WHISPER can detect a large and diverse attack vector with more than 99% accuracy at a reasonably low performance overhead.

INDEX TERMS Side-Channel Attacks (SCAs), Cryptography, Detection, Machine Learning, Security, Privacy.

I. INTRODUCTION

Information security is fast becoming a first-class design constraint in almost all domains of computing. Modern cryptographic algorithms are used to protect information at the software level. These algorithms are theoretically sound and require enormous computing power to break with brute-force. For instance, for a 128-bit AES key, it would take 5.4×10^{18} years to crack the AES using a computer capable of performing 10^6 decryption operations per μs [1]. However, recent research has shown that cryptosystems, including AES, can be compromised due to the vulnerabilities of the underlying hardware on which they run. Side-channel attacks

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

exploit such physical vulnerabilities by targeting the platforms on which these cryptosystems execute [2].SCAs can use a variety of physical parameters, *e.g.*, power consumption, electromagnetic radiation, memory accesses and timing patterns to extract secret keys/information [3], [4], [5], [6]. The baseline idea here is that the SCAs can exploit variations in these parameters during the execution of cryptosystems on a particular hardware and can determine the secret information used by cryptosystems based on the observed parameters. Cache-based side-channel attacks (CSCAs) are a special type of SCA, in which a malicious process deduces the secret information of a victim process by observing its use of caching hardware. Such attacks often rely on the presence of specialized instructions to manipulate the state of shared caches. The inherent features that any known CSCA exploits are cache timing and access patterns. In order to make an attack possible, an attacker usually needs to identify the victim's memory addresses by observing the shared caches. Further, an attacker would also have to determine if a particular memory access generated by the victim results in a cache hit or a miss. Generally, attackers observe the victim's memory accesses indirectly rather than directly. Thus, they change the usual utilization (with respect to a no-attack scenario) of underlying hardware resources, particularly caches. Such attacks can be prevented at various levels including at the system level, hardware level and application level [7]. At the system level, physical and logical isolation approaches exist [8]. At the hardware level, mitigation techniques are rather difficult due to the cost and complexity of their design. However, hardware solutions suggest having new secure caches, changes in prefetching policies and either randomization or complete removal of cache interference [9]. At the application level, the proposed countermeasures tend to target the source of information leakage and mitigate it [10]. The attacks are becoming sophisticated and stealthier [4], [7]. They overcome statically applied mitigation techniques. Therefore, on the one hand, protection against these CSCAs needs to be applied across the entire computing stake and, on the other hand, mitigation strategies must not reduce or remove the hard-earned performance benefits of computing systems. There is a niche for using detection-based protection, which would help apply mitigation only after successful detection of CSCAs.

This work addresses the problem of accurate and early detection of CSCAs at run-time. In this paper, we propose a machine learning system based on a runtime detection tool, called WHISPER, for CSCAs targeting Intel's x86 architecture. We demonstrate that intelligent performance monitoring of concurrent processes at hardware level, coupled with machine learning methods, can enable early detection of high precision and stealthier CSCAs. The state-of-the-art, discussed in Section II, suggests that some solutions exist based on machine learning for detection of CSCAs such as [11], [12], [13], [14], [15]. However, there are three major limitations in the prior work. The first is that the majority of machine learning models used in these solutions are trained to classify a specific attack, or a subset of attacks, belonging to any one category. Thus, when exposed to other CSCAs, these models have to be retrained. The second limitation is that, even after the machine learning model is retrained, it may not yield the same accuracy because different CSCAs exploit different cache vulnerabilities and the same model might simply not be capable of accurately classifying the changed behavior. And the third limitation is, prior works that are capable of detecting different attack techniques together are proved to have ill-suited evaluation criteria, *i.e.*, they do not account for performance overhead, detection speed, miss classification rates and load conditions simultaneously, for accurate and fast detection of attack behaviors at runtime. In practice, the system can be exposed to multiple attacks of different categories, in any temporal order and under any system load conditions. Thus, retraining or changing individual machine learning models may not be feasible, particularly for runtime detection tools. A more generic solution is needed that takes into account all (or most) attack techniques and, at the same time, well-suited evaluation criteria for detection.

In this paper, we use multiple machine learning models in an ensemble fashion in the WHISPER tool. The use of the ensemble model, instead of an individual machine learning model, is motivated by the fact that the proposed tool is designed to detect a large set of CSCAs. The ensemble model uses behavioral data of concurrent processes running on Intel's x86 architecture. These data are collected from different hardware events using HPCs, in near real-time, and are used as features. These data represent the pattern of memory accesses generated by data-dependent cryptographic operations that are being carried out by the underlying hardware. Since each CSCA generates a different interference with caches, the data being captured through HPCs at run-time can therefore lead to miss-classification for a single machine learning model. Instead an Ensemble model, incorporates multiple best-performing models and performs a majority-vote before classifying a given situation as Attack or No-Attack. It is thus capable of accurately detecting a larger set of attacks. Through extensive experiments, our goal is to evaluate the capability of the proposed tool to detect different variants of three state-of-the-art CSCAs, namely; Flush+Reload, Flush+Flush and Prime+Probe. The experiments are conducted to illustrate the proposed tool's capability of detecting almost all major known attack categories that are based on cache access patterns. The following are the main contributions of this work.

1) The paper proposes a run-time detection tool to detect CSCAs. The tool, called WHISPER, uses multiple machine learning models in an ensemble fashion and relies on the run-time profiling of concurrent processes that are collected directly through the hardware events using HPCs in near real time.
2) The paper demonstrates the capability of the proposed tool to detect access-driven side-channel attacks with reasonably high detection accuracy, high detection speed, low performance overhead and minimal false

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

positives and negatives.

3) For validation purposes, the paper presents experimental evaluation of the proposed tool against a large and diverse attack vector comprising both computational and storage attacks. Results for the detection of state-of-the-art CSCAs, such as Flush+Reload, Flush+Flush and Prime+Probe are presented and discussed. Results on the detection of computational attacks, such as Spectre and Meltdown are also presented to demonstrate the scalability of WHISPER tool.
4) We demonstrate that the WHISPER tool is resilient to noise generated by the system under various loads. To do so, we provide results under realistic system load conditions, *i.e.*, under No Load (NL), Average Load (AL) and Full Load (FL) conditions. These load conditions are achieved by concurrently running memory-intensive SPEC benchmarks on the system along with the cryptosystem and attacks.
5) We provide detailed discussion, supported with experimental data, about the selection of appropriate machine learning models and hardware events for run-time detection of CSCAs. Our results show that the proposed tool can also be used for other attacks (*i.e.*., Spectre and Meltdown attacks) without requiring any changes in the selected HPCs or retraining of classifiers.

The rest of the paper is organized as follows. Section II, provides the state-of-the-art on CSCAs, their mitigation and detection techniques. Section III discusses selected attacks as use-cases for evaluation of the proposed tool. Section IV presents the WHISPER tool, its components and design challenges associated with detection. Section V provides experimental evaluation using recent attacks from 3 CSCA categories. Section VI provides additional results on the detection of computational attacks to showcase the scalability of the WHISPER tool. Section VII discusses the results and Section VIII presents concluding remarks on our findings.

II. RELATED WORKS

In this section, our main aim is to provide a detailed background on the state-of-the-art related to CSCAs, their mitigation techniques and their proposed detection mechanisms.

A. STATE-OF-THE-ART ON CACHE SIDE-CHANNEL ATTACKS AND MITIGATION TECHNIQUES

CSCAs are a significant class of SCAs that exploit the execution-level details of cryptosystems by observing their interference with the cache hierarchy. The main sources of information leakage exploited by CSCAs are the memory access pattern of target cryptosystem and the access timing disparity in cache hierarchy. Many side- and covert-channel attacks have already been proposed [16]. For instance, some cache-based side-channel attacks are Flush+Reload [3], Flush+Flush [4], Prime+Probe [17], Evict & Time [5] and Evict & Reload [18]. Similarly, some covert-channel attacks include: Foreshadow [19] and May the Fourth Be With You [20]. More recently, new vulnerabilities have been discovered that exploit computational optimizations such as Out-of-Order and Speculative execution. Spectre [21] and Meltdown [22] attacks are the most recent examples of such attacks. Both Spectre and Meltdown are two-phase attacks and, in one of their phases, they require CSCAs to retrieve privileged information from caches. CSCAs are broadly divided into two classes, *i.e.*, time-driven and trace-driven attacks. Since cache hits and misses exhibit different timing for the same operation, this difference allows time-driven attacks to extract information on the secret key. Trace-driven attacks, on the other hand, try to access the cache lines that are being used by the victim by analyzing the cache state.

Multiple mitigation techniques have also been proposed against these CSCAs in the last decade. These techniques can be categorized into logical & physical isolation techniques, noise-based techniques, scheduler-based techniques and constant time techniques (referring to different cache levels in the cache hierarchy). For instance, logical physical isolation techniques include Cache Coloring [23], CloudRadar [24], STEALTHMEM [10], NewCache [25] and Hardware Partitioning [26]; noise-based techniques include fuzzy times [27], bystander workloads [28]; and scheduler-based techniques include obfuscation [28] and minimum timeslice [29]. Our Study of these mitigation solutions reveals that most techniques focus on only one specific vulnerability in the cache hierarchy. In practice, a system can be exposed to multiple attacks simultaneously or in any temporal order. Moreover, with relatively smarter attacks, these mitigation approaches can be bypassed. Moreover, researchers have discussed the fact that these attacks have not been completely patched to-date [7], [30]. Researchers have also debated that cryptographic code designers are far away from incorporating the appropriate counters to avoid cache leakages [31]. Since in this work, our focus is on the detection strategies of CSCAs, we provide more insight into detection mechanisms in the following and invite interested readers to explore further by following up references to mitigation techniques.

B. STATE-OF-THE-ART ON DETECTION MECHANISMS

SCA detection techniques are divided into two basic categories; signature-based and anomaly-based detection. Some techniques use a combined approach as well, *i.e.*, signature + anomaly-based detection. Signature-based techniques depend on signature of known SCAs. At run time, program execution is compared with an already generated signature and in the case of a match, an attack is detected. Such detection mechanisms show good accuracy for known attacks but suffer from low accuracy in the case of unknown or modified attacks [24]. Anomaly-based detection approaches generate a model for normal/benign applications. Any significant deviations from the model will be detected/considered as an attack. Anomaly-based detection mechanisms have the capability to detect unknown and modified attacks, but they can suffer from high false positives as it is difficult to include the behavior of every benign application. Also, benign applications can potentially resemble CSCAs due

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

to intensive memory usage [24]. There are some detection mechanisms that merge both anomaly and signature based detection mechanisms to achieve accurate results [32], [24].

1) Signature-based Detection

Allaf *et al.* [15] propose a mechanism to inspect Prime+Probe and Flush+Reload attacks targeting AES cryptosystem. Their mechanism uses ML models and HPCs but the work lacks the basic evaluation criteria required for detection. The proposed mechanism shows good accuracy under isolated conditions, where the attacker and victim are the only load on the system whereas, accuracy degrades under realistic load conditions. Moreover, the authors do not discuss the impact on overall performance caused by the said technique. Mushtaq *et al.* [12] proposed a signature-based run-time detection mechanism called the NIGHTs-WATCH. It comprises 3 linear machine learning classifiers, LDA, LR, and SVM, that work independently. The NIGHTs-WATCH takes different hardware events as input features under attack/no-attack scenarios. The authors claim a low performance overhead for a high detection accuracy and speed. This work is limited to the use of individual ML models trained and used for specific attacks. This means that models need to be retrained for each new attack to detect it. In a similar work, Mushtaq *et al.* in [13] used both linear and non-linear ML classifiers to detect different implementations of Prime+Probe attacks running on the AES cryptosystem.

In [33], Sabbagh *et al.* proposed a signature-based detection tool, SCADET, that detects Prime+Probe attacks. Instead of using HPCs, this approach uses high-level semantics and invariant patterns of attack, *i.e.*, I-cache, D-cache and LLC. Results show that SCADET provides very good accuracy, but no results on detection speed or on the performance overhead of the mechanism, which leaves the question of runtime adaptation. Authors report that, in some cases, the system provides false alarms under load conditions. Moreover, the trace analysis time in this approach is very long, meaning the solution is not suitable for runtime detection. Notable irregularities have also been reported when the trace exceeds a certain size.

Some of the signature-based detection mechanisms use threshold determination to detect CSCAs instead of machine learning [34], [35], [36]. One of the techniques in [34], named as HexPADS, utilizes the values of cache miss rates and page faults to detect an attack. HexPADS is able to detect cache template attacks [18] based on Flush+Reload, enhanced version of C5 attack [37] based on Prime+Probe, page fault attacks (CAIN [38]) and fault attacks based on Rowhammer [39]. However, the paper lacks a discussion on detection speed. Moreover, HexPADS assumes no variation in the system load, therefore, overhead may increase with realistic scenarios. Another similar threshold-based technique is presented in [35], which uses cache miss rate and data-TLB miss rate to recognize CSCAs. Authors reported results depicting successful detection of variants of Flush+Reload attacks. However, the authors mentioned neither performance overhead caused by this detection technique nor the detection speed. Raj and Dharanipragada [36] presented Pokerface to identify and mitigate CSCAs. Pokerface compares the memory bus bandwidth with a threshold level to detect Prime+Probe and Flush+Reload CSCAs. However, the authors do not provide a discussion of detection accuracy or mention the speed of the proposed mechanism. A threshold-based detection technique Deja-Vu [40] was introduced to detect attacks on programs guarded by SGX. This technique uses test benchmarks with a negligible overhead. However, the instrumentation required can increase the size of enclave binaries to a very large number and threshold determination is not always very sophisticated or provides a reliable approximation for detecting attacks.

2) Anomaly-based Detection

Recently, two new detection techniques that rely on anomaly-based detection have been proposed [41], [42]. CacheShield [41] is an anomaly-based detection mechanism for CSCAs on legacy software (victim applications) that monitors HPCs while using an unsupervised anomaly detection algorithm called Cumulative Sum Method (CUSUM). The CacheShield detects Flush+Reload, Flush+Flush and Prime+Probe attacks under load conditions and the results show that attacks were detected before the attack was 50% complete. Another anomaly-based detection mechanism is a semi supervised technique called the SpyDetector [42]. It has been proposed to detect CSCAs under variable load conditions using HPCs. The SpyDetector has been validated on Flush+Reload, Flush+Flush and Prime+Probe attacks running over RSA, AES and ECDSA cryptosystems. The mechanism performs well under variable load conditions on physical as well as on virtual setups, but the authors do not provide any results regarding detection speed and the overhead for reported accuracy.

3) Signature + Anomaly Based Detection

As discussed earlier, there are detection techniques that use a combination of both signature- and anomaly-based detection approaches [11], [24], [32]. Chiappetta *et al.* [11] proposed a machine learning based detection mechanism for Flush+Reload attack on AES and ECDSA cryptosystems. This work uses three approaches to detect the attack: correlation-based detection, anomaly detection and artificial neural networks-based detection. This detection mechanism offers good detection accuracy and speed but does not incur less overhead. Another signature and anomaly-based detection mechanism is CloudRadar [24], which correlates cryptographic execution of applications on virtual machines and the anomalous behavior of caches to detect CSCAs in cloud systems. The CloudRadar demonstrates its efficiency by performing an attack and once it is detected, VM migration is performed, which serves as a light-weight patch to cloud systems. The proposed mechanism is tested for Flush+Reload and Prime+Probe attacks and offers high accuracy and negligible overhead at a predetermined

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

sampling frequency and threshold. This approach does not take machine learning into account for analyzing variable behaviors and performs a threshold-based decision which is unreliable for different attack behaviors at runtime, where, the realistic load conditions can be noisy and attack behavior may vary from predetermined threshold based behavior. Although *CloudRadar* detects attacks with high accuracy in isolated conditions, no tests have been performed under realistic/noisy scenarios to measure the efficiency of the detection mechanism. Another three-step detection mechanism on cache and branch predictor based CSCAs is proposed in [32]. This method includes HPCs and machine learning models and uses three stages; detecting the anomaly, finding the class of anomaly and correlating malicious process with the victim. Their experiments show that this mechanism performs well and provides high accuracy under a predetermined sampling frequency and a specific threshold, but the detection speed and detection overhead of the proposed technique are not evaluated or discussed.

Based on our findings, the overall state-of-the-art on detection techniques are lacking in certain aspects that are crucial for making run-time detection an effective way to mitigate SCA attacks. For instance, most of the proposed techniques primarily focus on achieving higher detection accuracy and either ignore or insufficiently cater for other parameters like detection speed and performance overhead. Moreover, the attack surface is continuously expanding, but the proposed techniques target only selected attacks and do not try to cover a larger attack vector. We believe that, for run-time detection to be an effective way of mitigating SCAs, the detection tools need to cover a large attack vector while offering reasonably good detection accuracy, speed and overhead.

III. USE-CASES: SELECTED CACHE SIDE CHANNEL ATTACKS

As highlighted in Section I, we evaluated the proposed tool against a large and diverse attack vector to demonstrate its adaptability. Our primary focus, however, is on cache side-channel attacks. We selected six different CSCA implementations as use-cases for the validation of the WHISPER tool. These attacks cover three main categories of CSCAs, *i.e.*, Flush+Reload (F+R), Prime+Probe (P+P) and Flush+Flush (F+F). In the state-of-the-art, these selected use-case attacks often target a given cryptosystem to retrieve secret key information. Therefore, for the purpose of uniformity and demonstration, all the experimental results in this work are mainly shown for one cryptosystem, *i.e.*, AES. However, it is worth mentioning here that the WHISPER tool also works effectively in other cryptosystems, such as RSA. For the selected CSCA use-cases, we used two different versions of OpenSSL on which the attacks are demonstrated in the state-of-the-art. Table 1 details these use-cases. Later, in Section VI, we demonstrate that the WHISPER tool is also capable of detecting attacks that are independent of the cryptosystem, such as Spectre and Meltdown attacks.

To serve the community at large, we provide the source code and experimental data related to all these CSCAs at our Github repository [43], which can be freely accessed, used, distributed and reproduced.

TABLE 1: List of Selected CSCAs as Use-Cases

No.	Use-case CSCAs	OpenSSL Version	Key Recovery
1	Flush+Reload	0.9.7l/ 1.0.1f	Half Key
2	Flush+Reload	0.9.7l/ 1.0.1f	Full Key
3	Flush+Flush	0.9.7l/ 1.0.1f	Half Key
4	Flush+Flush	0.9.7l/ 1.0.1f	Full Key
5	Prime+Probe	0.9.7l/ 1.0.1f	Half Key
6	Prime+Probe	0.9.7l/ 1.0.1f	Full Key

A. FLUSH+RELOAD

Flush+Reload [3] is a trace-driven attack that relies on the presence of page sharing. State-of-the-art attacks have been performed using the Flush+Reload technique using RSA and AES cryptosystems [44], [6]. This technique has three major steps; first the attacker flushes the shared cache line using CLFLUSH instruction. In the second step, the attacker lets the victim execute and in the third step, the attacker reloads the cache line and measures the reload time. Reload time indicates if the cache line was of interest for the victim or not. Two different variants of this attack were implemented on the AES cryptosystem for our test cases, which are available in [6], [45] (full key, faster implementation), [46] (half key, slower implementation).

B. PRIME+PROBE

Prime+Probe attacks are in the category of trace-driven attacks that exploit last level shared caches across multiple cores. The principle of Prime+Probe has been used to develop various CSCAs such as those proposed in [28], [47], [5], [44], [48], [49]. The Prime+Probe technique involves two major steps; in the first step, the attackers prime the cache with their own data and let the victim execute, in the second step, the attackers probe the cache and measure the time needed to access their own primed data in cache. The difference in time will inform the attacker if any of the cache set has been accessed by the victim or not. The Prime+Probe technique has been used at different cache levels including L1-Data cache (L1-D) [48], L1-Instruction cache (L1-I) [50] and Last Level Cache (LLC) [51]. Two different variants of this attack technique are available; one implementation contains half key recovery with a slower implementation [46], whereas we implemented the Flush+Reload on AES from [6], [45] and modified it for faster and full key recovery on the same principle for Prime+Probe.

C. FLUSH+FLUSH

Flush+Flush [4] attacks also belong to trace-driven attack techniques, but replace the Reload step of Flush+Reload with a Flush step. Attacks rely on CLFLUSH instructions to perform the Flush step. This technique has two major steps;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

in the first step, the attacker flushes the cache line from the shared address space and lets the victim to operate normally (the victim may or may not access the flushed cache line). In the second step, the attacker flushes the same cache line and measures the time of flushing (the attacker determines if the victim has accessed that cache line or not during its normal execution). We implemented two different variants of this attack technique on the AES cryptosystem in our test cases and these are available on [4], [46] (this implementation targets the first round of encryption to recover half key which is rather slow). We also implemented Flush+Reload on the AES from [6], [45] and modified it for faster and full key recovery on the same principle as that used in Flush+Flush. The Flush+Flush attack is considered a stealthy, high-resolution and non-detectable CSCA. The Flush+Flush attack is also considered to be fast because unlike other CSCAs, it does not access any memory. Flush+Flush causes minimal cache hits and no cache miss at all due to constant flushes mechanism. Gruss *et al.* [4] claimed that the spy process in this attack can not be detected by monitoring cache behavior (hits, misses). Due to its working principle, a Flush+Flush attack causes more cache misses and memory accesses for the victim process due to constant flushing of its instruction/data.

We build our claim based on the fact that detection is required to indicate the presence or absence of intrusion from the victim's perspective. That is, for detection as a first step, it is important to identify when an intrusion is happening in the system in order to take the right protective measures. Then, the steps can be taken to identify which particular process in the system is the malicious one.

IV. WHISPER -A RUN-TIME SCA DETECTION TOOL

Having established the background and state-of-the-art in the previous sections, we now present the design details of our proposed CSCA detection tool, WHISPER. One of the distinguishing features of the WHISPER tool is that it is designed to work at runtime, *i.e.*, when the attack is actually happening, to detect and help the operating system protect itself against known CSCAs. The three challenges we address in designing the WHISPER tool are: 1) Detection tools usually approximate the whole system behavior which can increase the number of false positives and false negatives at runtime, 2) The detection process can slow down the overall execution of the cryptosystem, which can lead to a significant performance overhead while trying to achieve greater detection accuracy and 3) Detection can sometimes be very slow, resulting in late detection in the sense that the attacker has already completed up to 50% of its activity, for instance, secret key retrieval. In the literature, 50% is considered as a theoretical threshold for a successful attack [2], [5]. We considered all these design challenges as our evaluation metrics for the WHISPER tool.

The WHISPER tool does intelligent performance monitoring of concurrent processes with data collected at the hardware-level using hardware events and feeds them to selected machine learning models to perform early detection of high precision and stealthier CSCAs. The tool has two major components: 1) Selection of appropriate hardware events that will provide, at runtime, insight into the cache behavior while CSCAs take place and 2) Selection of appropriate machine learning methods that could perform binary classification of Attack vs No-Attack scenarios with high accuracy, high speed and minimum performance overhead.

We consider that the tool will operate under realistic system load conditions on commodity hardware. Therefore, we emulate the load conditions by running memory-intensive SPEC benchmarks on the system as background load. The load conditions are defined such that a No Load (NL) condition involves only victim and attacker processes running, an Average Load (AL) involves victim, attacker and any two SPEC benchmarks running, and a Full Load (FL) condition involves Victim, Attacker and any four SPEC benchmarks running in the background. In the state-of-the-art, it often happens that attacks are running in isolated conditions *i.e.*, attacker and victim are the only load. For instance, the attacks presented in [4], [5] and [3] have no load conditions. Therefore, to assume a realistic scenario, it is useful to validate the mechanism under different load conditions, which, in turn, validate the functioning of the tool in a realistic scenario. This is why we deliberately used SPEC benchmarks that are memory intensive and affect caches in terms of cache accesses, CPU cycles, cache hits and misses, *etc.*

In this section, we detail the methodology of the WHISPER tool and selection criteria for the two aforementioned components. We validate the tool experimentally on six different CSCAs and their variants. The CSCA use cases considered cover a large attack surface. In this work, we evaluate the efficacy of the proposed tool with these CSCAs being validated on different OpenSSL versions. We provide empirical evidence for the effectiveness of the proposed detection tool against these attacks and variants.

A. METHODOLOGY

Figure 1 is an abstract view of the methodology used in the WHISPER tool. We consider shared memory architecture as most known CSCAs target Intel's x86 based execution platforms. As illustrated, the tool collects behavioral data on concurrent processes at runtime using HPCs. As discussed in Section IV-B, these data comprise selected hardware events, which are fed to an Ensemble model. The WHISPER tool methodology consists of three distinct and significant phases, namely; 1) Run-time profiling, 2) Training machine learning models and 3) Classification & detection. In the following, we describe these phases in detail.

1) Run-time Profiling

The first phase of WHISPER's detection mechanism comprises runtime profiling of the victim's process *i.e.* target cryptosystem. In this phase, runtime samples from selected hardware events are collected using HPCs. Since we target access-driven CSCAs, we consider only the hardware events that are most affected by these attacks. Section IV-B provides

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

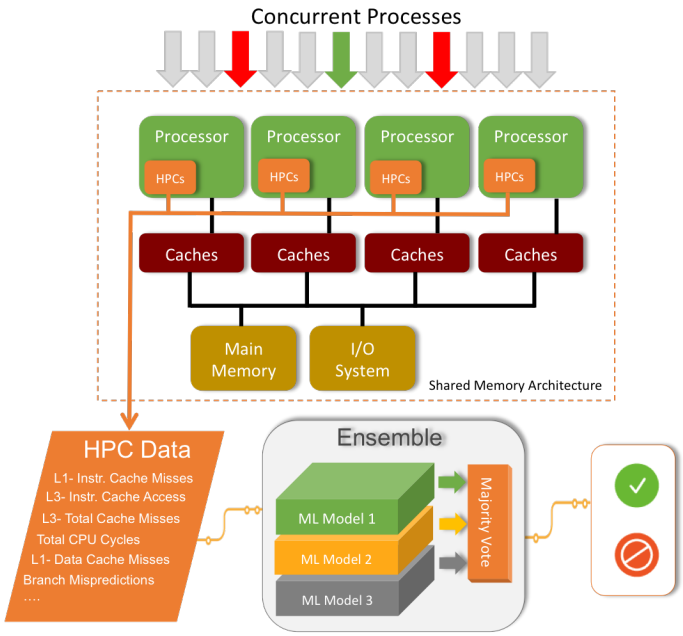


FIGURE 1: WHISPER Tool's Methodology-Abstract view

details on the selection of such events. Intel x86 processors [52] provide access to hundreds of hardware events through software APIs, which can reveal valuable information on the system.

WHISPER offers runtime profiling at variable sampling granularity. That is, the tool allows both the user and the system to adjust sampling granularity between fine-grain and coarse-grain levels, depending on the prevailing threat level. Sampling granularity is defined as the rate at which the data from HPCs are collected with respect to the completion of encryption. For example, Flush+Reload is a single encryption attack on RSA. Therefore, in this case, the sampling granularity would be after every $n-$ bits of encryption defined by the user. For attacks that require multiple encryptions to be completed, the sampling rate can be defined as $n-$ encryptions. For instance, Flush+Flush attack on AES requires between $350-400$ encryptions at least. Therefore, rather coarse-grain sampling would still be enough to detect this attack before its completion.

Sampling granularity has a major influence on the victim's performance as it varies the execution time of the victim's process and its shared libraries compared to normal execution. A higher sampling rate for hardware events would provide precise measurement of cache behavior, which would facilitate in faster detection speed. However, it will also slowdown the encryption/decryption process of the victim and increase the performance overhead. Therefore, sampling granularity implies a trade-off between detection speed and performance overhead. The experimental results in Section V show that our detection mechanism provides better detection speed at reasonably low performance cost, both at high and low sampling granularity. Once collected, these runtime profiles of the victim's process under variable system load conditions are used for training and cross validation of the machine learning models in the next phase. Post-training, the classifiers work with the same hardware events at runtime.

2) Training of machine learning models

The hardware events that are used to profile the target cryptosystem at runtime enable the tool to differentiate between attack and no-attack processes. We collected training data on one million samples evenly distributed among representative execution scenarios for the victim's process. For instance, the samples comprise an equal number of training data for attack and no-attack scenarios. Furthermore, each of these scenarios comprises an equal number of training data being collected under no load, average load and full load conditions. We then train the selected machine learning models with these labelled data. Our training data set is un-biased, *i.e.*, it does not favor one execution scenario over the other. For validation purposes, we apply K fold cross validation technique [53] for each individual model using training data to check its accuracy before deploying these models for runtime detection.

3) Classification & Detection

In the third and last phase, trained individual classifiers in the tool utilize runtime data originating from hardware events for classification and detection purpose. Based on training in the second phase, every model classifies the run-time data in two categories: Attack or No-Attack. A majority vote is then taken by the ensemble model on the individual decisions of selected machine learning models to decide whether the cryptosystem is under attack or not. Detection accuracy is based on how well the model is trained, whereas the speed of detection and performance is dependent on how fast we try to collect the samples, *i.e.*, sampling granularity. The rate of miss classifications is the measure of imperfections and inclinations of the predicted results and is defined as False Positives (FPs) and False Negatives (FNs). FPs is the case when a no-attack condition is detected as an attack and leads to loss of performance and loss of confidence. On the other hand, FNs is a condition in which an attack is detected as a no-attack, which is more of a security breach and involves loss of critical information. The results in Section V show how accurately and rapidly different models classify data samples at runtime in our proposed detection mechanism.

B. HARDWARE PERFORMANCE COUNTERS

Hardware performance counters (HPCs) are special purpose hardware registers present in almost all modern processor families. HPCs are basically used to monitor the performance of applications on architectural events (*e.g.*, cache misses/hits, CPU cycles, cache references, *etc.*) while applications are running on underlying hardware. Processors based on Intel's x86 architecture [52] provide access to hundreds of hardware events that can reveal valuable information of the system using HPCs. However, modifiable HPCs are limited in number. Therefore, few events can be monitored concurrently (ranging from 4 to 8 events). There

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

are many high-level libraries and APIs that can be used to configure and read HPCs, such as; PerfMon [54], OProfile [55], Perftool [56], Intel Vtune Analyzer [57] and PAPI [58], *etc.* As mentioned in Section II-B, many detection techniques use HPCs to detect different CSCAs. Selection of most appropriate and minimum number of hardware events that could help revealing the attack behavior remains an important challenge for detection tools.

1) Selection of HPCs

Many hardware events provide valuable information regarding normal vs abnormal behavior of running processes. For instance, Figure 2 shows some experimental results of selected hardware events (under NL conditions) that measure L1-Data Cache Misses (L1-DCM), L3-Total Cache Accesses (L-TCA) and L3-Total Cache Misses (L3-TCM) and Total Cycles for $100,000$ encryptions of the AES cryptosystem. Figure 2 shows the frequency of samples on the Y-axis and the magnitude of measured event on the X-axis. Results shown in green represent normal behavior of AES encryption algorithm running under no-attack, and results in red show AES running under Prime+Probe attack. Figure 2 shows that the magnitude of these events varies particularly (increases in this case) under an attack situation compared to in a no-attack (normal) situation, indicating that the events are particularly affected during the attack. Our detailed experiments with other cache-related hardware events show that they reveal very interesting information about CSCAs.

TABLE 2: Selected events related to CSCAs

Scope of Event	Hardware Event as Feature	Feature ID
L1 Caches	Data Cache Misses	L1-DCM
	Instruction Cache Misses	L1-ICM
	Total Cache Misses	L1-TCM
L2 Caches	Instruction Cache Accesses	L2-ICA
	Instruction Cache Misses	L2-ICM
	Total Cache Accesses	L2-TCA
	Total Cache Misses	L2-TCM
L3-Caches	Instruction Cache Accesses	L3-ICA
	Total Cache Accesses	L3-TCA
	Total Cache Misses	L3-TCM
System-wide	Total CPU Cycles	TOT_CYC
	Branch Miss-Predictions	BR_MSP

Since we target access-driven CSCAs, we consider only hardware events that are the most affected by these attacks. We performed experiments on a larger set of related hardware events (25+), including cache and system-wide counters and selected the 12 most significant events depicting the cache behaviors, as shown in Table 2. These events have also been used in previous studies for CSCA detection [59], [60], [12], [32]. Any of these events can be useful for detecting CSCAs relying on cache behaviors. Selection of events is

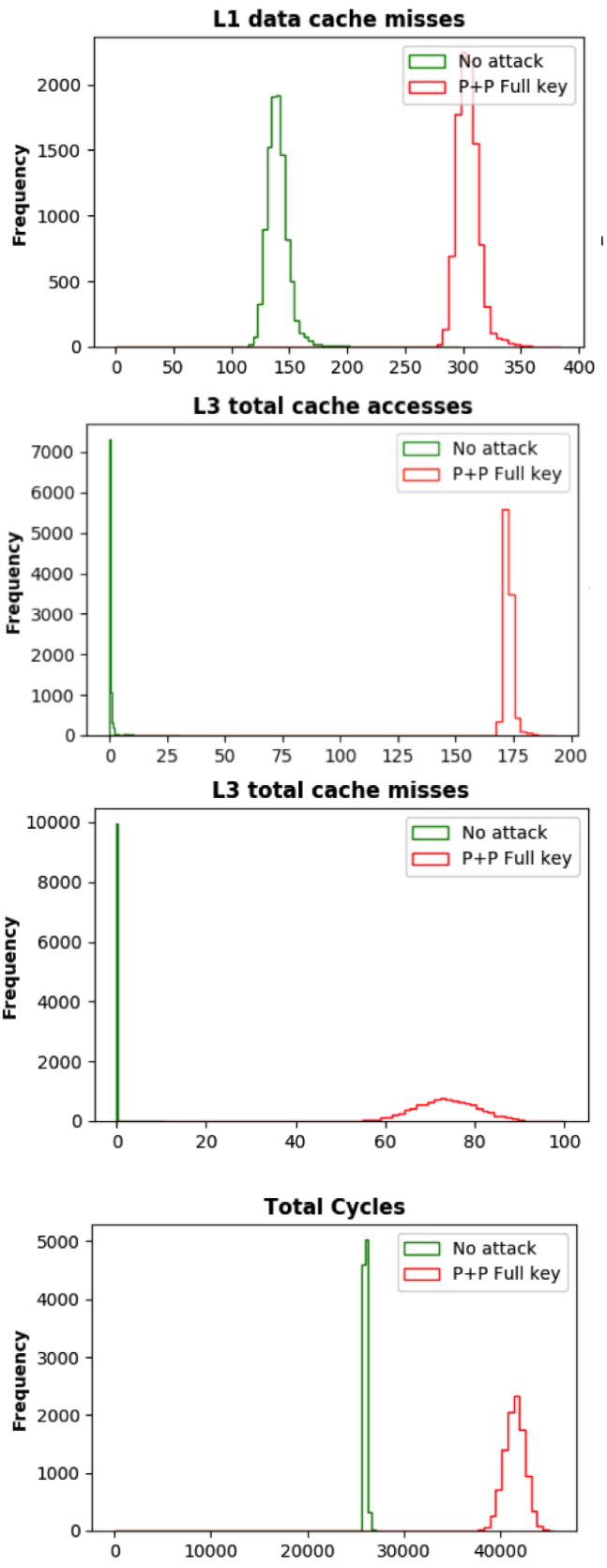


FIGURE 2: Results showing selected hardware events under no load condition for Prime+Probe attack on AES.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

TABLE 3: Selected events related to use-case CSCAs

Attack	Hardware Event as Feature	Feature ID
Flush+Reload	L1- Data Cache Misses	L1-DCM
	L3-Total Cache Accesses	L3-TCA
	L3-Total Cache Misses	L3-TCM
	Total CPU Cycles	TOT_CYC
Flush+Flush	L1- Data Cache Misses	L1-DCM
	L3-Total Cache Misses	L3-TCM
	L3-Total Cache Accesses	L3-TCA
	Total CPU Cycles	TOT_CYC
Prime+Probe	L1-Data Cache Misses	L1-DCM
	L3-Total Cache Accesses	L3-TCA
	L3-Total Cache Misses	L3-TCM
	Total CPU Cycles	TOT_CYC

strictly based on the following five arguments: 1) the relevance of events with respect to attack's behavior, 2) provision of precise and distinctive information on normal/abnormal behavior that does not overlap with any other set of events, 3) diversity and non-correlation among features that can be fed to machine learning models to enable confidant decision-making under realistic scenarios, 4) a minimum but sufficient set of events that does not cause huge performance overhead while sampling, and 5) minimum events that do not require multiplexing of hardware performance counters (as multiplexing can lead to non-deterministic and imprecise measurements at run time). Since every selected attack has its own peculiar characteristics and yet they affect caches in one way or the other, for the WHISPER tool, we chose the most suitable minimum number of hardware events that could maximize the understanding of targeted vulnerabilities in the cache behavior. To elaborate further, for instance, almost all cache-based side-channel or covert-channel attacks target the sharing property of the caching hardware (e.g., inclusive LLC) or use specialized instructions like *CLFLUSH* instruction. They increase the cache miss rate by continuously flushing selected cache lines, which also affects the total CPU time for the victim's process. Thus, we select the hardware events that are the most affected ones by the cache's behavior when system is under attack. As illustrated in Figure 2, we can see that attack behavior is mostly influenced by these 4 events; attacker creates a significant rise in L1 data cache misses and L3 total cache misses due to continuous flushing, the total number of access of cache increase and distinguishes normal/abnormal behaviors due to the number of total cycles increase as attacker is performing activity along with victim's execution. It is important to note that tweaking HPCs with relevance to attack behavior can provide a lot of information on system behavior to launch a detection mechanism.

Based on the CSCA characteristics reported in Section III, we selected the hardware events mentioned in Table 3 as being the best suited. For instance, figures 2 and 3 present experimental results of these selected hardware events for Prime+Probe attack under no load and full load conditions,

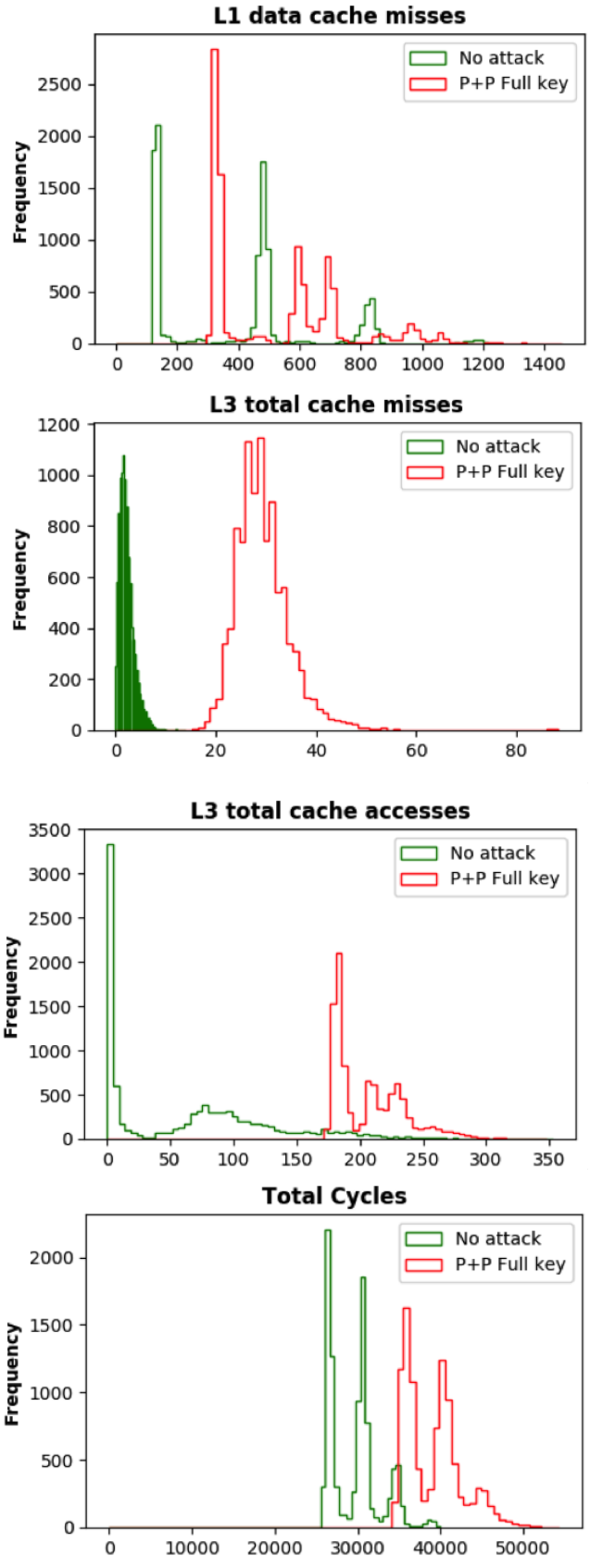


FIGURE 3: Results showing hardware events under Full Load conditions for Prime+Probe attack on AES.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

respectively. As depicted in these figures, each hardware event shows distinguishable features for attack and no-attack cases.

TABLE 4: List of Machine Learning Models for CSCA Detection (Non-exhaustive)

No.	Machine Learning Model	Category
1	Linear Regression (LR)	Linear
2	Linear Discriminant Analysis (LDA)	Linear
3	Support Vector Machine (SVM)	Linear
4	Quadratic Discriminant Analysis (QDA)	Non-linear
5	Random Forest (RF)	Non-linear
6	K-Nearest Neighbors (KNN)	Non-linear
7	Nearest Centroid	Linear
8	Naive Bayes	Linear
9	Perceptron	Linear
10	Decision Tree (DT)	Non-linear
11	Dummy	Non-linear
12	Neural Networks	Non-linear

C. MACHINE LEARNING MODELS

In this section, we discuss the rationale for selecting machine learning models for the WHISPER tool. In section IV-B, we discuss how hardware events can provide valuable information regarding the behavior of target processes and how the load conditions can affect their ability to provide distinguishable information. The difference between attack and no-attack scenarios is quite clear under No Load conditions as shown in Figure 2, which could easily be separated using a threshold. However, under a more realistic load condition such as Full Load, this situation worsens, as shown in Figure 3. Due to increased interference with caches, it becomes hard to separate an attack scenario from a no-attack scenario with simple threshold-based approaches.

Adding to the problem, in practice, a system can be exposed to multiple CSCAs simultaneously or in any temporal order, which would further increase the difficulty in distinguishing an attack scenario using data from hardware events. To explain this, we conducted experiments with six different attacks as use-cases (discussed in Section III). We collected data for selected hardware events under all load conditions for these six use-cases separately and then plotted them together as shown in Figure 4 for the full load case in order to illustrate the overlapping nature of data under different attacks. As can be seen in Figure 4, it is not easy to classify these HPC's data using simple threshold-based approaches.

For such a system, machine learning models can be helpful by appropriately learning the behavior of each CSCA using HPC data. However, any selected ML model is assumed to deal with such a data mix in order to separate an attack from a no-attack scenario, whereas our experiments show that not

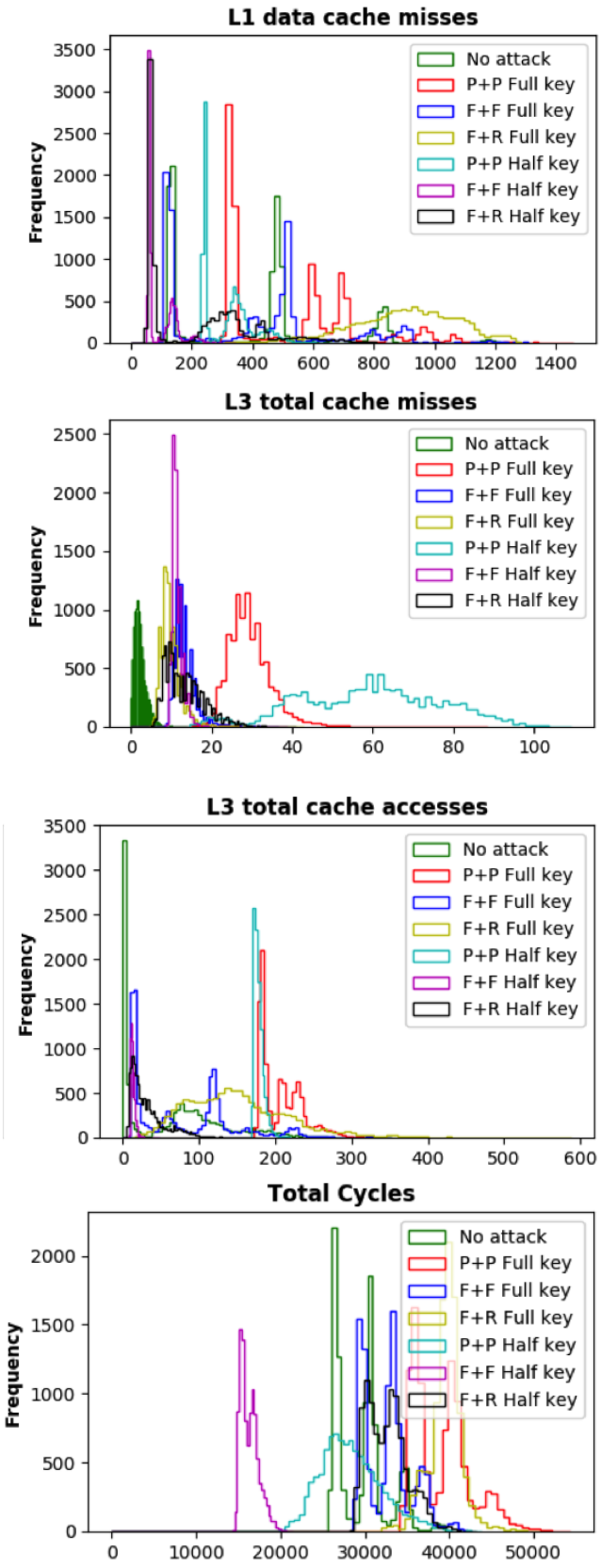


FIGURE 4: Results showing selected hardware events under Full Load conditions for six CSCAs on AES.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

all ML models yield acceptable results and it is important to understand the diversity of data affecting the caches in an attack/no-attack scenario for selection of ML models. To this end, in Section IV-C1, we discuss the behavior of mixed data collected under all six attacks that affects the caches. Analysis of these data provides an insight about the data mix and helps select appropriate ML models for the WHISPER tool. Based on the analysis in Section IV-C1, we then discuss the selection of ML models in Section IV-C2.

1) Dimensionality Reduction of Data

Before we present the results for different use-cases, it is important to have a look at the data that are being used for classification. Since the WHISPER tool is designed to detect multiple attacks at runtime, therefore, also for training purposes, the tool simultaneously takes into account the data from six attacks exhibiting different behaviors. It becomes very difficult for the ML models to perform behavioral analysis on such a mix of data. Figure 5 illustrates the data density plot between two selected hardware events, L1_DCM & L3_TCA in this case, under Full Load conditions for combined data of all attacks. These results show that the ML features are heavily correlated, which is the reason why not all linear models were able to perform well on these data. Other density plots between selected hardware events reveal similar information and are therefore omitted in favor of space.

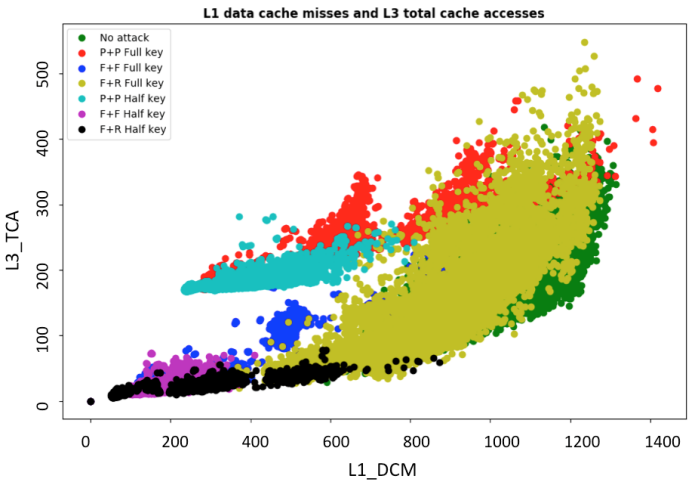


FIGURE 5: Results for data density between L1_DCM & L3_TCA under Full Load conditions -All attacks combined.

For the sake of clarity, we reduced the dimensionality of data for visualization using t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm [63]. Other tools can be used for data reduction including Principal Component Analysis (PCA) [64]. PCA is a mathematical technique but t-SNE is a probabilistic one. In order to represent high dimension data on low dimension, non-linear manifold, similar data points have to be represented close together, which can be more easily achieved by t-SNE rather than with PCA. As a result, in Figure 6, the data reduction can be seen from high scale to a good classification scale for Full Load conditions. This

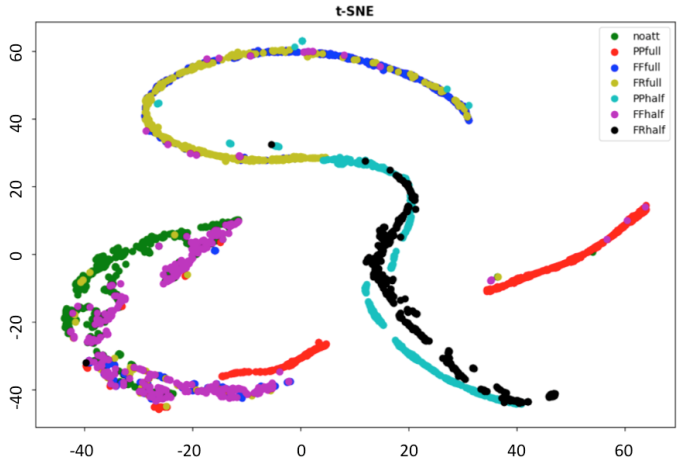


FIGURE 6: Results on data reduction and visualization using t-SNE algorithm under Full Load conditions -All attacks combined.

representation of data, on the one hand, provides us with the necessary information to select best-suited ML models for the tool and, on the other hand, provides a good insight into the cache behavior under multiple attacks. Figure 6 shows that, in reduced dimensions, the no-attack case is still distinguishable from the attack case if appropriate ML models are used.

2) Selection of Machine Learning Models

Machine learning models are broadly classified into two basic categories; linear and non-linear models. We experimented with a set of 12 distinguished ML models, among which, 6 models are linear and 6 models are non-linear as mentioned in Table 4. LDA is a statistical model used in Machine Learning to discriminate classes by finding a linear combination of the input features such that it provides the best separation between classes. SVM is a non-probabilistic supervised Machine Learning model also known as maximum margin classifier. SVM learns and chooses an optimal hyperplane through the training data in such a way that its distance from the nearest training data points is maximum on each side. The hyperplane then characterizes the new input data points. QDA is a statistical, non-linear, supervised model used in Machine learning. QDA is somewhat similar to LDA in the context of assumptions, except for one assumption of the same co-variance matrix between the features. QDA finds a non-linear combination of features such that it provides the best separation between classes. RF is an ensemble learning method that is used for classification and regression purposes. It is mostly used for over-fitting of training sets. It is robust to the inclusion of irrelevant features and produces inspectable models. It explodes in the form of a tree and is able to grow deeply following highly irregular patterns. KNN is a non-parametric statistical approach used in pattern recognition and for supervised classification in Machine Learning. KNN classifies an incoming data point by assigning it the same label as that of its maximum K-nearest training data points

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

label. This algorithm is computation and memory intensive. The Nearest Centroid is a parametric supervised classifier used in ML. It calculates the distance of the new input data point from the mean of the training data of each class and then assigns the label to the new input data point in the class whose calculated distance was smallest. Naive Bayes is a probabilistic supervised ML classifier with a strong assumption that the features are independent of each other. It calculates the probability of an incoming data point in each class. The new data point gets the label of the class whose probability is greater. Perceptron is like the neuron in human brain. It is a linear supervised ML approach which represents the simplest neural network. It learns some weights for future use from the training data and then predicts the class of the new incoming data point by calculating the weighted sum of these features. DT is a tree like model of decisions and their possible consequences. It is one of the models that contains conditional control statements. In decision analysis, DT mainly helps to carve a strategy to reach certain goals. Dummy Model is a naive approach that can be used for classification. It assigns the label of the most frequent class to the new input data. Neural Networks, also known as multilayer perceptrons, are composed of multiple perceptron hidden layers. Feed forward and backward propagation techniques can be used for error reduction. Detailed explanations about these models are available in [53], [61], [62], [63]. We performed experiments with well-known basic ML models as they exhibit low overhead at runtime with considerable accuracy. This list is non-exhaustive and does not imply rejection of the use of other ML models.

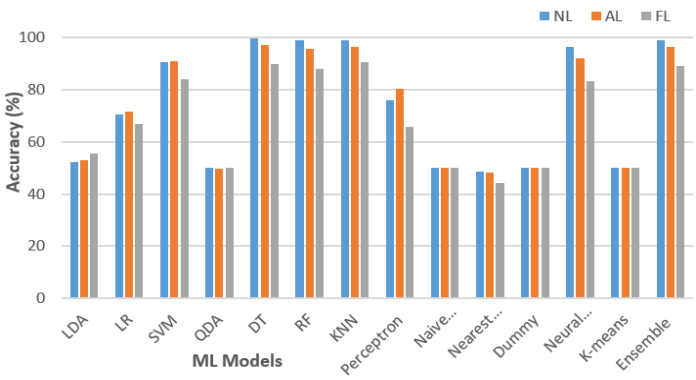


FIGURE 7: Accuracy Comparison of ML models for 6 Attacks

Since WHISPER is a runtime CSCA detection tool, we apply stringent criteria for selection of ML models that best suit our design constraints, *i.e.*, classification accuracy, implementation feasibility for run-time detection, performance overhead, distribution of error (false positives and false negatives) and detection speed. Detection accuracy is the most important criterion for the tool. Therefore, we test the detection accuracy for all 12 models with a training data set collected for all six attacks running on the system. We then compare these models on the rest of the design constraints. Figure 7 illustrates the detection accuracy of all 12 models that were tested against six attacks with variable load conditions (No Load, Average Load and Full Load). Figure 7 shows that linear models, including LDA, LR, Naive Bayes, Nearest Centroid and Perceptron do not perform very well on detection accuracy while classifying the HPC data of six attacks. This is because of the significant data overlap. Similar pattern can also be observed in some of the non-linear models, including QDA, Dummy and K-means. Based on the detection accuracy alone, it leaves us with a choice of SVM, DT, RF, KNN and Neural Networks. However, detection accuracy, the most important parameter, is not the only one to consider when deploying a high-speed run-time CSCA detection tool. The other very important parameter to examine while comparing ML models is their implementation feasibility. Likewise, ML models should be able to quickly provide their decision while keeping their performance overhead minimum. According to these criteria, although the KNN model shows good detection accuracy, it uses all training data points at runtime to infer decision. Thus, KNN involves significant implementation complexity, which can lead to high performance and storage overheads. Neural network is a model that performs well in terms of detection accuracy as shown in Figure 7. It uses a forward and backward propagation technique that causes the problem of over-fitting due to which it backtracks all events for error reduction and eventually increases performance overhead. That is why we did not use neural networks in the WHISPER tool, but it could be a useful selection for sophisticated attacks in future. For the moment, our problem can be resolved using models that provide less overhead than Neural Networks. That is why we selected the remaining three models with a good performance, *i.e.*, DT, RF and SVM. All these models perform consistently, offer good accuracy and ease of implementation while providing negligible performance overhead (as demonstrated by experimental evidence). The WHISPER tool uses these three ML models to demonstrate its effectiveness. However, the tool is scalable and can include/exclude other ML models with one-time training phase, as explained in Section IV-A.

Our experiments revealed that, although all three selected ML models perform reasonably well when used individually, they still exhibit high rate of false positives and false negatives because of the variation in HPC data captured under different attacks. Therefore, the WHISPER tool uses all three models in an ensemble fashion where the final decision on detection is taken through a majority-vote among the best-performing ML models for runtime input data. The results in Figure 7 show that the ensemble of models shows consistent accuracy for diverse data coming from six different attacks at runtime.

D. IMPLEMENTATION OF DETECTION MODULE

Before presenting the experimental evaluation, in this section, we briefly present the implementation details of the WHISPER tool. Algorithm 1 presents an abstract level pseu-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

docode for WHISPER's detection module. As illustrated, the module takes as input the sampling granularity for hardware events (`SamplingGranularity`), which can be either user-defined or adjusted at runtime. By default, the sampling granularity is set to fine grain, *i.e.*, 10 AES encryptions. The sampling in WHISPER is user-defined and can be changed based on the type and the level of threat at runtime. Moreover, different attacks require observation of a different number of encryptions in order to be successful. For instance, there are certain attacks that require observation of only a single encryption to complete (e.g., Flush+Reload on RSA-1024 bits [3]) while other attacks require observtion multiple encryptions to complete (*e.g.*, Flush+Flush on AES [4]). Therefore, in the case of Flush+Reload on RSA, WHISPER can set the sampling granularity at the rate of every 10 bits, whereas 10 encryptions for Flush+Flush on AES require observation of roughly $350 - 400$ encryptions to complete the attack. Sampling granularity can always be tuned by the user/system to effectively control the performance overhead depending on the type and level of threat at runtime. Another input is the total number of iterations for which we tested the module (`MaxIterations`). The number of iterations varies for each attack as discussed in Section V. Lines $1 - 3$ show that a victim's (encryption) process is initialized, the detection module is activated once and the hardware events are set around the victim's process considering it as the ROI (Region of Interest). For the overall selected number of iterations, the module activates detection after every 10 or 100 encryptions in the case of AES cryptosystem, depending on what granularity has been specified through `SamplingGranularity` (lines $4-6$). Once activated, the detection module collects the data from hardware events (line 7) and feeds them as features to selected binary classifiers (line 8) in order to perform individual voting. Based on these individual votes, the module generates a report after taking a majority vote on the results (line 9). Detection is then deactivated (line 10) and if the report is `True` then an attack is reported (lines $11 - 12$). Otherwise, the victim's process continues to execute uninterrupted.

Algorithm 1: Run-time Detection Module

```
   Input: SamplingGranularity, MaxIterations
   Initialization:
   events← ∅, votes ← ∅
   report← False, VictimProcess ← NIL
1  VictimProcess← Get_Encryption_Process()
2  Activate_Detection(VictimProcess)
3  Set_Hardware_Events(VictimProcess)
4  for i ← 1 to MaxIterations 1 do
5      if i mod SamplingGranularity == 0 then
6          Activate_Detection()
7          events ← Read_Hardware_Events()
8          votes ← ML_Classifiers(events)
9          report ← Majority_Voting(votes)
10         Sleep_Detection()
11         if report == True then
               /* Attack Detected          */
12             return 1
   /* No-attack detected !                 */
13 return 0
```

V. EXPERIMENTS AND DISCUSSION

We evaluate the WHISPER tool under stringent design constraints. To do so, we create three experimental case studies to detect Prime+Probe, Flush+Reload and Flush+Flush attacks, respectively, on AES cryptosystem. In each case, we evaluate the performance of our tool under variable load conditions, *i.e.*, No Load (NL), Average Load (AL) and Full Load (FL) conditions, as explained in Section I.

A. SYSTEM MODEL

We demonstrate the efficiency of the WHISPER tool on Intel's core $i7-4770$ CPU running on Linux Ubuntu 16.04.1 with Kernel $4.13.0 - 37$ at 3.40-GHz. Our threat model consists of access-driven CSCAs, which produce information leakage over the entire cache hierarchy (L1, L2 & LLC) of Intel x86 architecture. We take same-core and cross-core CSCAs into account for the purpose of detection. We consider that the Operating System (OS) is not compromised in our system model (*i.e.*, it is part of the trusted computing base). The PAPI (Performance Application Programming Interface) library is used to access hardware events using HPCs from the Intel Core $i7$ machine. It provides machine and operating system independent access to HPCs. Any of over 100 preset events can be counted through either a simple high level programming interface or a more complete low level interface of PAPI. For experimentation, the training has been performed offline with 1-Million samples of unbiased data of attack and no-attack samples. For run-time inference, we analyzed the results by running the attacks $100,000$ times. In the following, we present our experimental results using 3 case studies. Additionally, for comparative analysis, we provide results using individual ML models running separately as well as in an ensemble fashion.

B. CASE STUDY-I: DETECTING PRIME+PROBE

Our first case study provides experimental evaluation of the detection of two implementations of a Prime+Probe attack targeting the AES cryptosystem.

1) Detection Accuracy

Detection accuracy is the primary indicator to judge the effectiveness of any detection tool. In all our case studies, we use percentage accuracy to demonstrate the validity of trained machine learning models. We use unbiased samples in the training and validation data, *i.e.*, samples for attack and no-attack cases are equal.

Tables 5 & 6 show our experimental results for individual ML models (RF, DT, SVM) and Ensemble, respectively, for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

two different implementations of the Prime+Probe attack. These results illustrate the variation in aforementioned metrics under different load conditions. All three ML models individually provide very high and consistent detection accuracy under No Load, Average Load and Full Load conditions, *i.e.*, between 92.67–99.99% for Impl.1 (Table 5) and between 97.73–99.99% for Impl.2 (Table 6). Evidently, the Ensemble model used by the WHISPER tool also performs very well and provides a detection accuracy ranging between 97.62–99.99% for Impl.1 (Table 5) and 96.94–99.77% for Impl.2 (Table 6), respectively. The results of Ensemble model are particularly interesting under Average Load and Full Load conditions where individual models might not always perform consistently. To support these results, we also provide the run-time behavior of hardware events under different load conditions. For instance, Figure 8 shows the behavior of hardware events for Prime+Probe attack Impl.1 under Full Load conditions and Figure 2 shows the same events for Prime+Probe attack Impl.2 under No Load condition. Figures 2 & 8 illustrate that, under Prime+Probe attack, the hardware events offer distinguishable behavior for attack and no-attack scenarios under all load conditions, which is why the detection accuracy for all ML models remains very high.

2) Detection Speed

Detection speed is another important criterion for the evaluation of any run-time intrusion detection tool. Detection speed is an indirect reflection of how aggressively a detection tool profiles the victim's process behavior (through hardware events in this case) and provides its decision. Detection speed also affects the resulting performance overhead of the tool as it is a trade-off between how fast an intrusion can be detected versus how much overhead the detection process would cost. According to the literature, a Prime+Probe attack would require AES cryptosystem to perform at least $4,800$ encryptions for Impl.1 and $50,000$ encryptions for Impl.2 [4], [6] in order to be successful. Therefore, the percentage of encryptions being performed before the WHISPER tool raises a flag, defines the detection speed with respect to attack completion. For instance, for Prime+Probe attack Impl.1, if the tool raises a flag before 4800 encryptions of AES are performed, then the tool is said to be capable of detecting a Prime+Probe attack on AES within 10% of attack completion. Please note that the detection speed is determined in a time-independent manner. Also, theoretically, it is considered enough for an attacker to deduce 50% of the secret key, as the rest of the key can be acquired using reverse engineering techniques [2], [5]. Therefore, it is safe to detect an attack before it can complete at most 50% by itself. For the WHISPER tool, we considered this the upper bound on detection speed.

Our experimental results in Tables 5 & 6 show that the WHISPER tool is capable of detecting Prime+Probe attack Impl.1 and Impl.2 within 0.21% and 0.02% of completion, respectively. This result implies that the WHISPER tool detects within the first 10 encryptions out of 4800 and $50,000$ encryptions required by Impl.1 and Impl.2, respectively. This speed is achieved under variable load conditions with fine-grain sampling frequency. Fine-grain is the highest profiling granularity used in these experiments in which the tool samples hardware events after every 10 encryptions. We also tested the tool with coarse-grain sampling granularity, as shown in Tables 7 & 8. Coarse-grain profiling would mean sampling hardware events after every 100 encryptions. Tables 7 & 8 show that the resulting accuracy remains almost the same or is further improved in some cases, while the system overhead decreases drastically compared to fine-grain sampling.

Results in Tables 7 & 8 reveal that the tool is still capable of achieving detection accuracy comparable to that of fine-grain detection, while the performance overhead is reduced by significantly large margins. For instance, in Tables 6 & 8

TABLE 5: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.1*: half-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.21	0.01	0.00	6.59
	AL	99.76	0.21	0.24	0.00	
	FL	95.00	0.21	5.00	0.00	
SVM	**NL**	99.99	0.21	0.01	0.00	7.83
	AL	99.82	0.21	0.18	0.00	
	FL	94.92	0.21	5.08	0.00	
RF	**NL**	99.99	0.21	0.01	0.00	11.3
	AL	99.72	0.21	0.28	0.00	
	FL	92.67	0.21	7.33	0.00	
Ensemble	**NL**	99.99	0.21	0.01	0.00	8.03
	AL	99.77	0.21	0.23	0.00	
	FL	97.62	0.21	2.37	0.01	

TABLE 6: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.2*: full-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.02	0.01	0.00	7.45
	AL	98.53	0.02	1.47	0.00	
	FL	98.54	0.02	1.46	0.00	
SVM	**NL**	99.99	0.02	0.01	0.00	6.75
	AL	98.50	0.02	1.50	0.00	
	FL	98.61	0.02	1.38	0.02	
RF	**NL**	99.99	0.02	0.01	0.00	9.34
	AL	97.90	0.02	2.10	0.00	
	FL	97.73	0.02	2.27	0.00	
Ensemble	**NL**	99.77	0.02	0.23	0.00	8.20
	AL	96.94	0.02	3.6	0.00	
	FL	99.09	0.02	0.91	0.00	

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

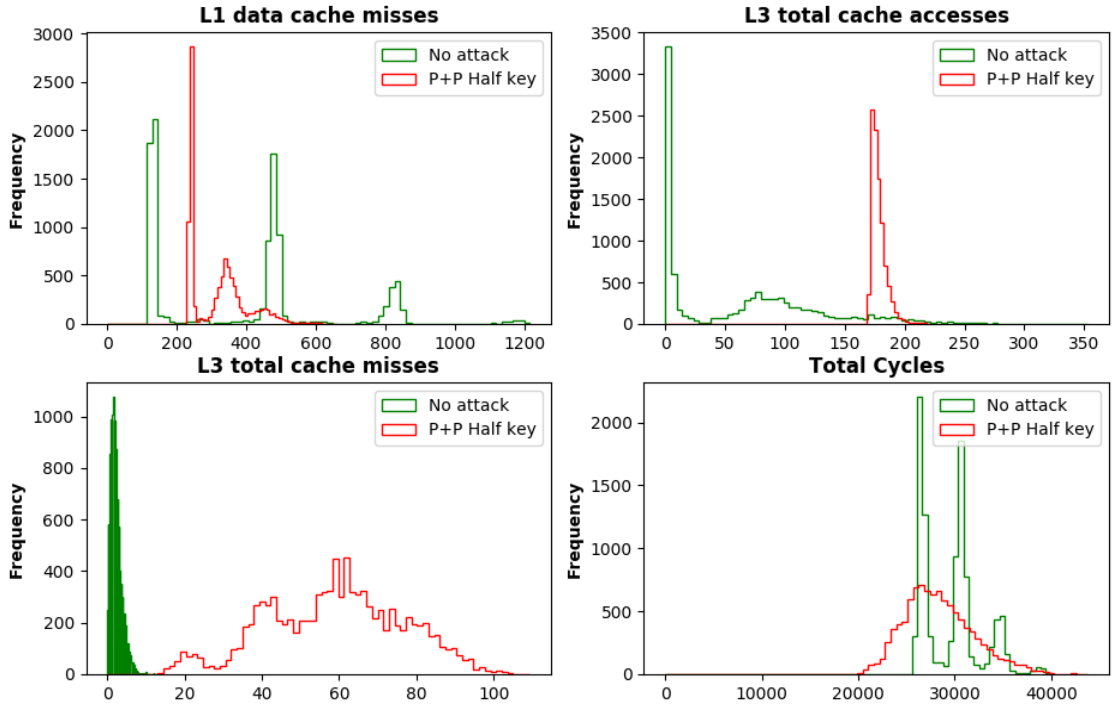


FIGURE 8: Run-time behavior of selected hardware events under Full Load conditions for Prime+Probe *Impl.1*

with Prime+Probe attack Impl.2, the performance overhead for the Ensemble model decreased from 8.20% to 1.03% –that is, by a factor of roughly 8! Similarly, the misclassification rate for FPs and FNs also decreases due to coarse-grain detection where, in most cases, FNs are always zero or negligible. Detection speed would naturally go down by a small margin in the case of coarse-grain detection as we sample hardware events only after every 100 encryptions.

TABLE 7: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.1*: half-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	2.08	0.01	0.00	0.30
	AL	99.82	2.08	1.47	0.18	
	FL	98.18	2.08	1.82	0.00	
SVM	**NL**	100	2.08	0.00	0.00	0.15
	AL	97.51	2.08	2.49	0.00	
	FL	98.72	2.08	1.28	0.00	
RF	**NL**	99.99	2.08	0.01	0.00	2.99
	AL	97.20	2.08	2.80	0.00	
	FL	97.70	2.08	2.30	0.00	
Ensemble	**NL**	99.99	2.08	0.01	0.00	3.01
	AL	97.48	2.08	2.52	0.00	
	FL	97.93	2.08	2.07	0.01	

TABLE 8: Results using individual and Ensemble ML models for detection of Prime+Probe (*Impl.2*: full-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.19	0.01	0.00	4.5
	AL	99.71	0.19	0.29	0.00	
	FL	98.19	0.19	1.81	0.00	
SVM	**NL**	99.99	0.19	0.00	0.01	0.82
	AL	99.32	0.19	0.66	0.01	
	FL	98.87	0.19	1.13	0.00	
RF	**NL**	99.99	0.19	0.01	0.00	1.71
	AL	99.21	0.19	0.79	0.00	
	FL	97.90	0.19	2.10	0.00	
Ensemble	**NL**	99.99	0.19	0.01	0.00	4.55
	AL	99.29	0.19	2.52	0.71	
	FL	98.19	0.19	1.81	0.00	

3) Confusion Matrix

Confusion matrix provides a prognosis of results by representing the number of correct and incorrect predictions by the ML models as False Positives (FP) and False Negatives (FN). Ideally, any error in the detection is not desired. The presence of FNs in the system are indicative of a direct security breach, whereas, the presence of FPs would generate a false alarm, thus leading to performance degradation in the system. From the point of view of security, having a bounded

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

amount of FPs compared to FNs is still less damaging to the system in the sense that FNs would be a breach of security while FPs would cause degraded performance. For comparative analysis, we look into the FPs and FNs generated by each individual model compared to the Ensemble model. As shown in Tables 5 & 7, for Prime+Probe attack Impl.1, all models offer very high accuracy, which naturally leads to a very low number of FPs & FNs. For instance, if we analyze the miss-classifications performed by these models under variable load conditions, the SVM model misclassifies 5.08% for fine-grain detection and 2.49% for coarse-grain detection, the DT model misclassifies 5.0% for fine-grain detection and 1.82% for coarse-grain detection and RF model misclassifies between 7.33% for fine-grain detection and 2.80% for coarse-grain detection, respectively. These are the maximum misclassification results. Compared to all these individual models, the Ensemble misclassifies 2.37% at fine-grain detection and 2.52% at coarse-grain detection in the worst case. A very important observation here is that, while the overall error in classification is already very low, misclassification only concerns False Positives. No False Negatives are generated by the tool, except for 0.01% of FN by the Ensemble model under FL conditions for fine-grain detection, which is negligible. For coarse-grain detection, the misclassification rate is almost always 0% for FP & FN.

We report similar results for Prime+Probe attack Impl.2 as shown in Tables 6 & 8. Here, SVM model miss-classifies between 1.50% for fine-grain detection and 1.13% for coarse-grain detection, DT model misclassifies between 1.47% for fine-grain detection and 1.81% at coarse-grain detection and RF model misclassifies between 2.27% for fine-grain detection and 2.10% for coarse-grain detection, respectively, in the worst-case. Compared to all these individual models, the Ensemble misclassifies between 3.6% for fine-grain detection and 2.52% for coarse-grain detection in the worst-case. For Prime+Probe attack Impl.1, the Ensemble model performs well for fine-grain detection and further reduces the misclassification rate for coarse-grain detection. For Impl.2, at coarse-grain detection, there are only 0.71% FNs in Average Load condition but in all other cases it implies 0.00% FNs.

4) Performance Overhead

The performance overhead of the detection tools becomes a particularly important design parameter in the case of runtime detection. Moreover, the adaptability and scalability of the tool also depends on its runtime performance overhead. As discussed briefly in Section V-B2, the detection granularity determines how fast the hardware events are profiled by a detection tool to make effective decisions at run-time. This granularity aggressively impacts the performance overhead as fine granularity would imply more time spent in profiling. We measure performance overhead as a percentage of slowdown experienced by the AES cryptosystem when detection is being enabled. Our experiments with selected ML models reveal that the performance overhead of the WHISPER tool is low, specifically at coarse-grained detection. Tables 5 & 6 show that the AES cryptosystem experiences a maximum slowdown of 11.3% and 9.3% for Impl.1 and Impl.2, respectively, at fine-grain detection granularity for the RF model. A coarse-grain sampling frequency of 100 encryptions for hardware events reduces the performance overhead to a maximum of 2.99% and 4.5% for Impl.1 and Impl.2, respectively, as shown in Tables 7 & 8.

C. CASE STUDY-II: DETECTING FLUSH+RELOAD

Our second case study concerns experimental evaluation of the detection of two implementations of Flush+Reload attack targeting AES cryptosystem.

1) Detection Accuracy

Although a Flush+Reload attack is considered a high-resolution CSCA, the WHISPER tool also demonstrates very high detection accuracy for this attack case study. Tables 9 & 10 show the experimental results of our individual ML models (RF, DT, SVM) and Ensemble, respectively, for two different implementations of Flush+Reload attack. Like the results of our first case study against Prime+Probe attack, all three ML models individually provide very high and consistent detection accuracy under No Load, Average Load and Full Load conditions, *i.e.*, between 97.17–100.00% for Impl.1 (Table 9) and between 98.52–99.99% for Impl.2 (Table 10). Similarly, the Ensemble model used by the WHISPER tool also performs very well and provides a detection accuracy ranging between 98.92–99.99% for Impl.1 (Table 9) and 98.37–99.99% for Impl.2 (Table 10). As shown in Tables 9 & 10, the selected ML models for the tool perform consistently even under Average Load and Full Load conditions against Flush+Reload attack.

The runtime behavior of hardware events gives more insight into these results. For instance, Figure 9 shows the behavior of hardware events for Flush+Reload attack Impl.1 under the No Load condition and Figure 10 shows the same events for Flush+Reload attack Impl.2 under the Full Load condition. These figures show that, under a Flush+Reload attack, the hardware events offer distinguishable behavior for attack and no-attack scenarios.

2) Detection Speed

Flush+Reload attack of the AES cryptosystem requires 250 encryptions for Impl.1 and $50,000$ encryptions for Impl.2 to successfully extract the secret key. Our experimental results, shown in Tables 9 & 10, show that the WHISPER tool is capable of detecting Flush+Reload attack Impl.1 and Impl.2 within 4.00% and 0.02% of their completion, respectively, *i.e.*, within the first 10 encryptions out of 250 and $50,000$ required by the Flush+Reload attack Impl.1 and Impl.2, respectively. Tables 11 & 12 show the results for coarse-grain sampling, where the detection accuracy remains more or less the same, while detection speed is reduced to 40% and 0.2% for Impl.1 and Impl.2, respectively. As illustrated in Tables Tables 11 & 12, the main advantage of using coarse-grain detection remains the significant decrease in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

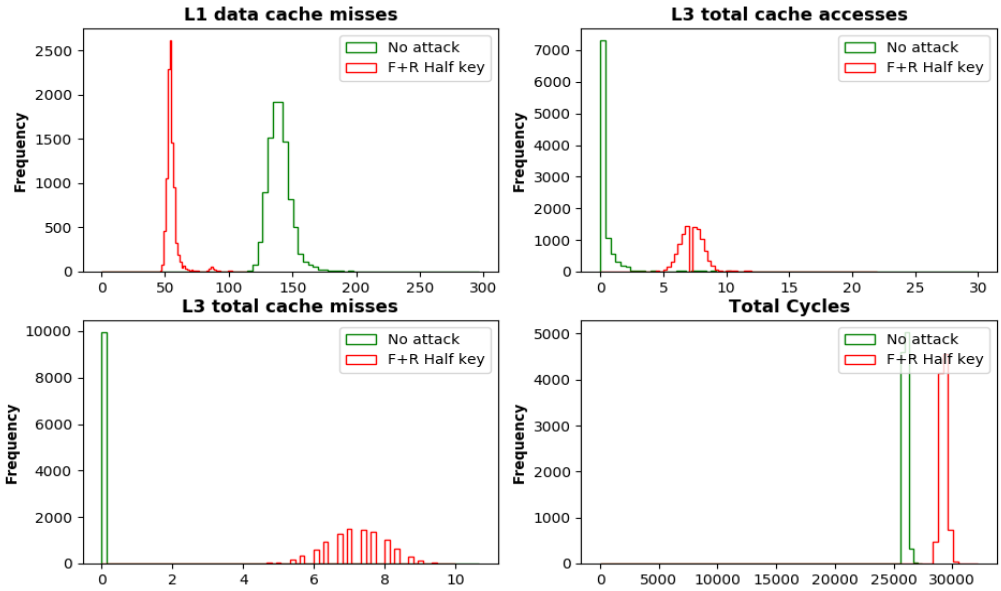


FIGURE 9: Run-time behavior of selected hardware events under No Load conditions for Flush+Reload *Impl.1*

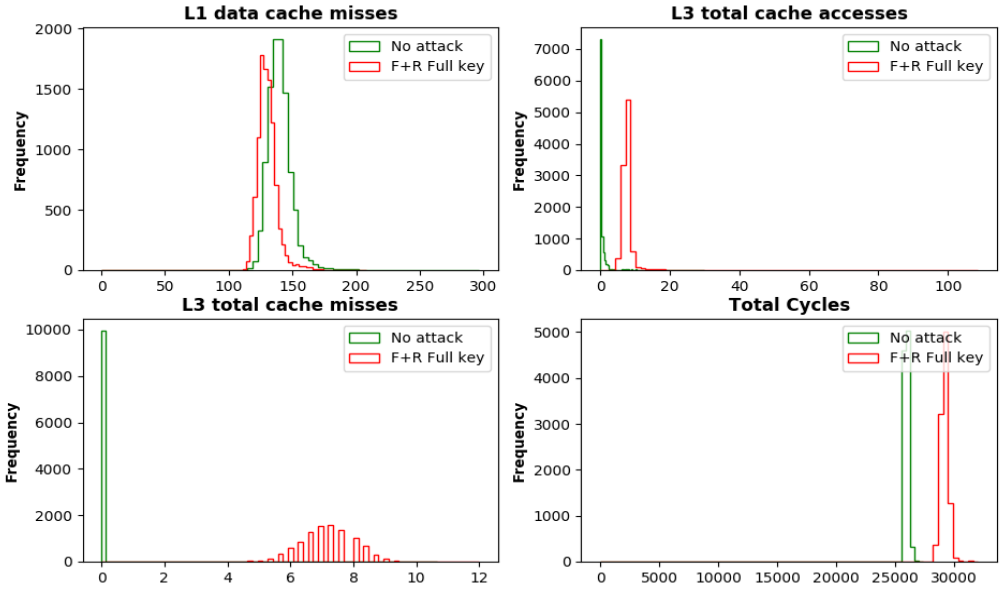


FIGURE 10: Run-time behavior of selected hardware events under Full Load conditions for Flush+Reload *Impl.2*

system overhead compared to fine-grain sampling. Similarly, the misclassification rate in terms of FPs and FNs is also reduced in many cases due to coarse-grain detection. The results show that the percentage of FNs is almost always zero or negligible. This explains why the WHISPER tool works efficiently for coarse-grain detection with a further decrease in system overhead and reasonable detection speed.

One important point to note here and in most of the upcoming results is the constant detection speed. We tested individual as well as Ensemble models with fixed sampling (fine-grain or coarse-grain). Therefore, the attack is detected at a constant speed for both individual and Ensemble models. Moreover, in some cases, the attack detection speed varies slightly due to variable load conditions, but the difference is negligible.

3) Confusion Matrix

We analyzed the misclassification error rate for the WHISPER tool w.r.t. Flush+Reload attack on the same pattern as we did for the Prime+Probe attack. In the Flush+Reload attack Impl.1 (with fine-grain and coarse-grain detection), as shown in Tables 9 & 11, the SVM model misclassifies between 0.86% and 1.53%, DT model misclassifies between 1.0% and 1.77% and RF model misclassifies between 2.83% and 2.17% for fine-grain and coarse-grain

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

TABLE 9: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.1*: half-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	4.00	0.01	0.00	10.8
	AL	99.71	4.00	0.29	0.00	
	FL	99.00	4.00	1.00	0.01	
SVM	**NL**	100	4.00	0.00	0.00	11.1
	AL	99.75	4.00	0.25	0.00	
	FL	99.14	4.00	0.86	0.00	
RF	**NL**	99.98	4.00	0.02	0.00	12.3
	AL	99.57	4.00	0.43	0.00	
	FL	97.17	4.00	2.83	0.00	
Ensemble	**NL**	99.99	4.00	0.01	0.00	11.2
	AL	99.68	4.00	0.32	0.00	
	FL	98.92	4.00	1.08	0.00	

TABLE 10: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.2*: full-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	0.02	0.01	0.00	11.17
	AL	99.44	0.02	0.56	0.00	
	FL	98.91	0.02	1.09	0.00	
SVM	**NL**	99.99	0.02	0.01	0.00	9.78
	AL	99.45	0.02	0.50	0.05	
	FL	95.92	0.02	0.70	3.38	
RF	**NL**	99.99	0.02	0.01	0.00	13.25
	AL	99.05	0.02	0.95	0.00	
	FL	98.52	0.02	1.47	0.01	
Ensemble	**NL**	99.99	0.02	0.01	0.00	8.27
	AL	98.37	0.02	0.63	0.00	
	FL	98.99	0.02	1.01	0.01	

TABLE 11: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.1*: half-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.99	40	0.01	0.00	2.00
	AL	99.86	40	0.13	0.02	
	FL	98.05	40	1.77	0.18	
SVM	**NL**	100	40	0.00	0.00	1.3
	AL	98.47	40	1.53	0.00	
	FL	98.82	40	1.18	0.00	
RF	**NL**	99.99	40	0.01	0.00	1.89
	AL	98.28	40	1.72	0.00	
	FL	97.83	40	2.17	0.00	
Ensemble	**NL**	99.99	40	0.01	0.00	2.19
	AL	98.44	40	1.56	0.00	
	FL	98.09	40	1.91	0.00	

TABLE 12: Results using individual and Ensemble ML models for detection of Flush+Reload (*Impl.2*: full-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.96	0.2	0.02	0.02	0.1
	AL	95.83	0.2	0.36	3.81	
	FL	98.30	0.2	1.70	0.00	
SVM	**NL**	100	0.2	0.00	0.00	0.41
	AL	98.44	0.2	1.56	0.00	
	FL	90.00	0.2	0.99	0.00	
RF	**NL**	100	0.2	0.00	0.00	2.0
	AL	98.20	0.2	1.80	0.00	
	FL	98.41	0.2	1.59	0.00	
Ensemble	**NL**	100	0.2	0.00	0.00	2.10
	AL	98.37	0.2	1.63	0.00	
	FL	98.60	0.2	1.40	0.00	

detection, respectively, in the worst case. Compared to all these individual models, the Ensemble model misclassifies between 1.08% and 1.01% of fine-grain and coarse-grain detection, respectively, in the worst case. The overall error in classification is also very low in this case and the misclassification mainly constitutes False Positives. It can be seen in Table 10 that Ensemble achieves 0.01% of FNs even in cases where individual models do not exhibit any FNs. This is because individual models, compared to Ensemble, are average results of different iterations and sometimes vary due to different runtime conditions. No False Negative is generated by the tool except in the DT model that provides 0.01% in Table 9 and $0.02\% - -0.18\%$ in Table 11. Similar results for Flush+Reload attack Impl.2 are shown in Tables 10 & 12. Here,the SVM model misclassifies between 0.70% and 1.56%, the DT model misclassifies between 1.09% and 1.70% and the RF model misclassifies between 1.47% and 1.80% only in fine-grain and coarse-grain detection, respectively, in the worst case under variable load conditions. The Ensemble model misclassifies between 1.01% and 1.63% in the worst case. The Ensemble model provides only 0.01% FNs under Full Load conditions, as shown in Table 10 and provides no FN, as shown in Table 12 for Flush+Reload Impl.2. If we compare all the results of misclassifications from No Load to Full Load conditions, it can be observed that coarse-grain detection helped reduce FPs and FNs in many cases. This is due to the fact that HPCs are inherently imprecise and non-deterministic, as discussed in Section IV-B1, which might lead to erroneous results in the case of fine-grain (rapid) sampling. A coarse-grain (slow) sampling

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

would allow hardware events to mature their measured values before being read.

4) Performance Overhead

As discussed earlier, performance overhead is more related to the implementation and sampling granularity of hardware events. Tables 9 & 10 show that the AES cryptosystem experiences a maximum slowdown of 12.3% and 13.25% while detecting Impl1. and Impl2. of Flush+Reload attack, respectively, under fine-grain detection granularity. This overhead is remarkably reduced to a maximum of 2% for both implementations of Flush+Reload attack in the case of coarse-grain detection.

D. CASE STUDY-III: DETECTING FLUSH+FLUSH

Our third and last case study provides experimental evaluation of the detection of two implementations of Flush+Flush attack targeting AES cryptosystem. Flush+Flush is a stealth and high-resolution attack that targets LLC except that it is stealthier than Flush+Reload and Prime+Probe attacks. Unlike other CSCAs, the stealth nature of Flush+Flush does not make any memory accesses. Thus, as a running process itself, it causes no cache misses and the number of cache hits are reduced to minimum due to constant cache flushing. Authors in [4] claimed to detect their own attack using hardware events in Linux $Perf\_event\_open$ interface. They documented that the attack (spy process) is non-detectable using 24 hardware events available with Linux syscall interface. Contrary to the claim in the above mentioned paper, we observed that the spy process might not be detectable on one hand, but that, on the other hand, the victim's process is affected by the constant high speed flushing. We build our argument around the fact that detection mechanism should indicate the presence or absence of intrusion from the victim's perspective. Specifically identifying the malicious process is often not required to apply protection. If, and when, an intrusion is detected in the victim's process, the OS can still take preventive measures like complete isolation or execute a critical section of the victim's process, etc. This work practically demonstrates that along with training of other CSCAs, WHISPER can detect stealth nature attacks like Flush+Flush. We achieved considerably high detection accuracy for Flush+Flush attack.

1) Detection Accuracy

In this case study, we demonstrate that Flush+Flush is not only detectable but also detectable with relatively very high detection accuracy using the WHISPER tool. Tables 13 & 14 show our experimental results for individual ML models (RF, DT, SVM) and Ensemble, respectively, for two different Flush+Flush attacks on similar patterns as shown in the other two case studies. In this case as well, all three ML models provide high and consistent detection accuracy when used as individual models under No Load, Average Load and Full Load conditions. As shown, the accuracy ranges between 71.84–99.98% for Impl.1 (Table 13) and between 72.86–99.56% for Impl.2 (Table 14). The Ensemble model performs better than individual ML models in this case and provides a very good detection accuracy ranging between 94.68–99.97% for Impl.1 (Table 13) and 94.82–97.71% for Impl.2 (Table 14). Please note that, in this case study, the use of the Ensemble model instead of individual models has a clear advantage. This is due to the stealthy nature of the Flush+Flush attack, which is not easily detectable by individual models under all load conditions. Thus, a major vote helps achieve the best possible accuracy. As shown in Tables 13 & 14, individual ML models show significant variations under load conditions against Flush+Flush attacks but the Ensemble model performs consistently even under Average Load and Full Load conditions. It can be seen that detection accuracy under Full Load conditions reaches 94% in both cases due to integrating three models in which one model provides poor accuracy, whereas, the other two models are between $94 - 97\%$ and $95 - 99\%$. This shows that the Flush+Flush attack is stealthy in nature and hard to detect and we rely on the Ensemble model rather than choosing individual models for detection. For instance, in this case, the SVM predicts with low accuracy. The run-time behavior of hardware events gives more insight for these results. For instance, Figure 13 shows the behavior of hardware events for Flush+Flush attack Impl.1 under No Load condition and Figure 12 shows the same events for Flush+Flush attack Impl.2 under Full Load condition. These figures show that, under a Flush+Flush attack, the hardware events offer much less distinguishable behavior for attack and no-attack scenarios compared to the other two use-cases, which causes the accuracy of ML models to vary. Nevertheless, our tool demonstrates that it is capable of precisely capturing this variation and provides high detection accuracy for stealthier attacks as well.

TABLE 13: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.1*: half-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.98	2.50	0.01	0.01	10.0
	AL	99.89	2.50	0.11	0.00	
	FL	97.77	2.50	0.81	1.43	
SVM	**NL**	71.84	2.50	0.01	28.14	14.5
	AL	84.52	2.50	0.13	15.35	
	FL	88.44	2.50	2.71	8.85	
RF	**NL**	99.94	2.50	0.03	0.03	12.74
	AL	99.73	2.50	0.26	0.01	
	FL	94.71	2.50	1.44	3.85	
Ensemble	**NL**	99.97	2.50	0.01	0.02	11.17
	AL	99.80	2.50	0.17	0.03	
	FL	94.68	2.50	4.36	0.96	

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

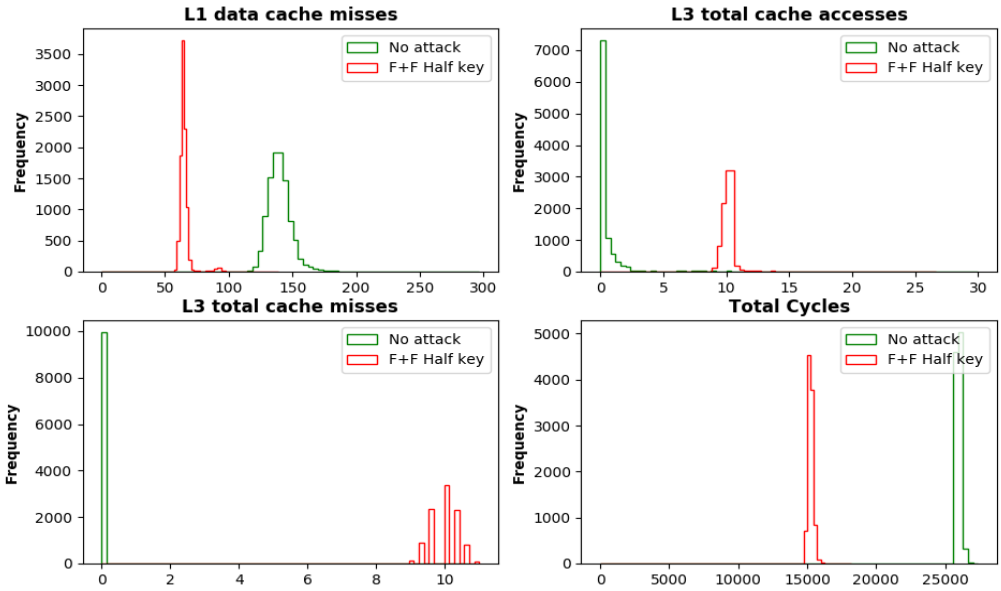


FIGURE 11: Run-time behavior of selected hardware events under No Load conditions for Flush+Flush *Impl.1*

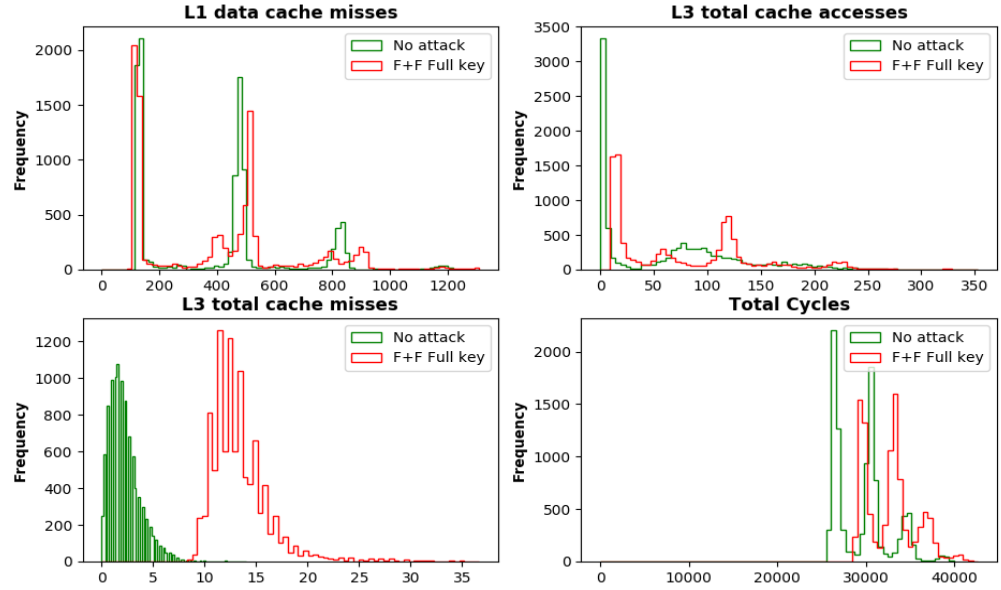


FIGURE 12: Run-time behavior of selected hardware events under Full Load conditions for Flush+Flush *Impl.2*

2) Detection Speed

A Flush+Flush attack on AES cryptosystem requires 350–400 encryptions for Impl.1 and $50,000$ encryptions for Impl.2 to extract the secret key. Our experimental results, shown in Tables 13 & 14, show that the WHISPER tool is capable of detecting Flush+Flush attack Impl.1 and Impl.2 within 2.5% and $0.02 - 0.04\%$ of their completion, respectively. That is, within the first $10 - 20$ encryptions out of $350 - 400$ and $50,000$ encryptions required by Flush+Flush attack Impl.1 & Impl.2, respectively. Like the first two case studies, this speed is achieved under variable load conditions under fine-grain detection granularity. Tables 15 & 16 show results for coarse-grain sampling, where the speed is reduced to 25% and 0.4% for Impl.1 and Impl.2, respectively, in the worst case. In terms of performance overhead, we observe a similar decreasing pattern as in other case studies in both implementations. With a Flush+Flush attack, the misclassification rate is generally higher than with the other two attacks. This is mainly due to the stealthy nature of this particular attack, which makes the models to misclassify more often. Here, we only analyze the reduction in the misclassification rate between fine-grain and coarse-grain detection scenarios

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

TABLE 14: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.2*: full-key recovery) on AES at fine-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.56	0.02	0.01	1.43	12.5
	AL	99.12	0.02	0.51	0.37	
	FL	97.84	0.02	1.54	0.62	
SVM	**NL**	72.86	0.02	0.02	27.12	11.5
	AL	87.44	0.04	0.50	12.06	
	FL	79.53	0.02	1.58	18.89	
RF	**NL**	98.16	0.02	0.08	1.76	14.78
	AL	98.80	0.02	0.95	0.25	
	FL	95.16	0.02	2.76	2.08	
Ensemble	**NL**	97.07	0.02	0.01	2.92	18.1
	AL	95.71	0.02	4.08	0.21	
	FL	94.82	0.02	1.76	3.42	

TABLE 16: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.2*: full-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	88.01	0.2	0.02	11.97	2.9
	AL	93.73	0.2	0.28	5.99	
	FL	90.31	0.2	1.80	7.89	
SVM	**NL**	74.07	0.4	0.00	25.93	4.13
	AL	90.68	0.2	2.19	7.12	
	FL	83.14	0.4	4.87	12	
RF	**NL**	87.32	0.4	0.03	12.65	5.46
	AL	92.71	0.2	2.48	4.81	
	FL	92.64	0.4	2.78	4.58	
Ensemble	**NL**	87.28	0.4	0.03	12.69	5.82
	AL	92.58	0.2	2.22	5.20	
	FL	91.72	0.4	2.47	5.81	

that are linked with detection speed. Overall, the misclassification rate decreases in the case of coarse-grain detection. Our results show that the magnitude of both FPs and FNs is reduced with coarse-grain sampling.

TABLE 15: Results using individual and Ensemble ML models for detection of Flush+Flush (*Impl.1*: half-key recovery) on AES at coarse-grain sampling.

ML Model	Load	Accuracy (%)	Speed (%)	FP (%)	FN (%)	Overhead (%)
DT	**NL**	99.97	25	0.02	0.00	0.2
	AL	99.79	25	0.20	0.00	
	FL	96.65	25	2.48	0.88	
SVM	**NL**	71.99	25	0.00	28.01	2.74
	AL	93.58	25	0.94	5.49	
	FL	91.43	25	1.50	7.07	
RF	**NL**	98.79	25	0.01	1.20	1.72
	AL	95.37	25	1.19	3.45	
	FL	96.07	25	2.88	1.05	
Ensemble	**NL**	98.79	25	0.01	1.20	2.96
	AL	95.54	25	1.01	3.45	
	FL	96.18	25	2.48	1.34	

3) Confusion Matrix

For a Flush+Flush attack, we analyze the misclassification error rate based on the same pattern as for Prime+Probe and Flush+Reload attacks. However, in this case, our findings are different. Tables 13 & 15 show our results concerning the error distribution as a percentage of FPs and FNs for Flush+Flush attack Impl.1 and Tables 14 & 16 show similar results for Impl.2. In case of Flush+Flush Impl.1, our experiments yield that individual models miss-classify with a significant high rate. For instance, SVM miss-classifies between 2.71%–1.50% as FPs and 28.14%–28.01% as FNs. The DT model miss-classifies between 0.81%–2.48% as FPs and 1.43%–0.88% as FNs. Similarly, the RF model miss-classifies between 1.44%–2.88% as FPs and 3.85%–3.45% as FNs in the worst-case. The Ensemble model, under the similar conditions, miss-classifies between 4.36%–2.48% as FPs and 0.96%–1.34% as FNs in the worst-case. Similar pattern can be found in case of Flush+Flush attack Impl.2 in Tables 14 & 16.

These results show that, for a given cryptosystem, depending on the target attack type and load conditions, some models may respond differently than others when used stand alone thus leading to much higher error rates in certain conditions. The use of an Ensemble model helps to maintain consistency in detection despite these variations and eliminates the impact of an individual model's error rate on overall accuracy under specific conditions. For instance, in this case, the effect of the SVM model error rate in detecting Flush+Flush attack has been eliminated thanks to the use of Ensemble by the WHISPER tool. The overall error in classification for the Ensemble model remains low and the miss-classification mainly constitutes False Positives. There is a small fraction of False Negatives, between 0.96%–1.34%, generated by Ensemble which is attributed to the stealthy nature of Flush+Flush attack. Yet, the overall distribution of error is well-managed by the Ensemble model.

4) Performance Overhead

Tables 13 & 14 show that the AES cryptosystem experiences a maximum slowdown of 14.5% and 14.7% while detecting Flush+Flush attack Impl.1 and Impl.2, respectively, under fine-grain detection granularity. This overhead is significantly reduced to maximum of 2.7% and 5.4% under coarse-grain detection for Impl.1 and Impl.2, respectively, as indicated in Tables 15 & 16. Overall, the performance

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

overhead is comparatively large for a Flush+Flush attack compared to Prime+Probe and Flush+Reload attacks. This variation mainly originates from the implementation of the attack, which could have different make-span and thus lead to additional (or fewer) iterations of the victim's process, causing the detection module to execute more often.

VI. SCALABILITY OF WHISPER -THE CASE OF COMPUTATIONAL ATTACKS

Section V has provided a detailed experimental evaluation of the WHISPER tool against a large set of known side-channel attacks. In this section, we demonstrate that the proposed tool is also capable of detecting a diverse attack vector, which is not necessarily known a priori, i.e., for which the tool is not trained beforehand. We consider both a trivial scenario in which the tool uses its training on 02 CSCAs (Flush+Flush and Prime+Probe) to detect another CSCA (Flush+Reload), as well a non-trivial scenario by training WHISPER tool with a set of CSCAs (i.e., Flush+Reload, Flush+Flush and Prime+probe attacks) and try to detect other attacks (i.e., Spectre [21] and Meltdown [22] attacks), which are computational attacks and unknown to the tool. The later evaluation scenario inherently covers the trivial case. However, do provide results for both scenarios in Section VI-A.

Attacks like Spectre [21] and Meltdown [22] exploit out-of-order and speculative execution techniques that are available in almost all modern processors to maximize the utilization of execution units in CPU cores. Unlike Flush+Reload and Flush+Flush, these attacks are independent of the cryptosys-tem and can retrieve data from the victim's address space without having access privilege. This section briefly elaborates the scalability of the WHISPER tool by detecting both Spectre and Meltdown attacks. We demonstrate the scalability under two scenarios. In the first scenario, no modification is performed to WHISPER, i.e., no new features (hardware or software events) are added and no retraining of the ML models is performed. In the second scenario, we show that adding new features to WHISPER that are specifically related to the computational part and retraining the ML models can yield even better results. In the following, first we briefly explain the attack vector. In modern processors, when the control flow of the application depends on the result of a preceding instruction, the processor can predict the most likely path of the program and speculatively execute the next instructions. Depending on the size of the reorder buffer, speculative execution can run several hundreds of instructions ahead. In practice, it is known that speculative execution can lead to the incorrect execution of a program, but the CPU is designed to revert the results of incorrect speculative executions by simply not committing such results. Therefore, these errors were assumed to be safe prior to the discovery of Meltdown and Spectre attacks. However, it turns out that not all side effects of speculative execution are revertible and some previously leaked information, for instance, cache contents, can survive the revision of the CPU state. The Spectre and Meltdown attacks exploit this flawed behaviour by recovering the leaked information from the cache using CSCAs like Flush+Reload and Prime+Probe *etc*. Although Spectre and Meltdown attacks exploit computational optimization techniques like out-of-order and speculative execution, they still use CSCAs like Flush+Reload and Flush+Flush to potentially retrieve information from the caches.

The Spectre vulnerability exploits hardware features, such as branch prediction units, to reveal secret data. Speculative execution in CPUs, which results from branch mispredictions, leaves observable effects in the cache. Spectre vulnerability is verified as being effective on Intel, AMD and ARM processors. A Spectre attack is executed in two distinct phases: a branch instruction mistraining phase followed by a CSCA. Spectre attacks are reported to have many variants. However, the baseline vulnerability exploited by all variants is the mistraining of branch instructions, be it a branch direction predictor or Branch Target Buffer (BTB).

Similarly, Meltdown vulnerability exploits out-of-order and speculative execution. Out-of-order memory look-ups also leave noticeable effects in the cache, which are exploited by this particular attack. Meltdown vulnerability affects Intel processors and, according to ARM, some of their processors are also affected. Meltdown is also a two-phase attack that, in the first phase, bypasses memory isolation by unprivileged out-of-order execution and, in the second phase, performs CSCA to retrieve information from the caches that is being brought in but not committed due to exception. Meltdown generates segmentation faults or invalid page faults while it bypasses the memory isolation. Our experimental results show that Meltdown generates significantly more page faults than other processes.

Meltdown relies on a vulnerability specific to Intel and ARM processors and can be mitigated by implementating KAISER [21] in operating systems, whereas Spectre applies to vastly more CPU architectures and cannot be mitigated as effectively. To the best of our knowledge, no research work has reported on the detection of Meltdown attacks prior to this work. Some research work has, however, reported the detection of a Spectre attack and its variants [64]. Further details on how Spectre and Meltdown attacks work are provided in [21] and [22], respectively.

For our experiments, we use Perf and PAPI libraries to extract events related to Meltdown and Spectre attacks, respectively. Although Spectre attack have multiple variants we use one of the variants proposed in the original work on Spectre attack [21]. Since both attacks are independent of the cryptosystem, their detection therefore requires monitoring of all concurrent processes. We wrap the events around all active processes to sample the features at a coarse-grain frequency of 100 ms and demonstrate the detection results. Thus, the performance overhead in this case comprises the sampling of events for all processes and are therefore slightly higher (Tables 20 & 21).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

A. SCENARIO 01: WITHOUT MODIFICATIONS IN THE WHISPER TOOL

It is pertinent to mention here that, in this case, the tool uses exactly the same HPCs and machine learning models without being retrained. That is, the tool has used its previous one-time training on cache-based SCAs to detect other attacks. At first, we have performed experiments with a trivial case in which, we have performed cross validation of the tool by training it with two CSCAs (Flush+Flush and Prime+Probe) and evaluated against an unknown CSCA (Flush+Reload), i.e., without training the tool for Flush+Reload attack. Table 17 shows that the detection accuracy is very high as anticipated intuitively. There were also no false negatives reported in this case.

TABLE 17: Results on the detection of Flush+Reload Attack using the WHISPER tool with training on only 02 attacks (Flush+Flush and Prime+Probe)

ML Model	Load	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	98.08	100	1.92	0.0	0.51
	AL	96.435	100	3.565	0.00	
	FL	92.77	100	7.23	0.00	
SVM	NL	98.80	100	1.20	0.00	0.77
	AL	96.35	100	3.65	0.00	
	FL	94.12	100	5.88	0.00	
RF	NL	98.48	100	1.52	0.00	0.46
	AL	95.35	100	4.65	0.00	
	FL	89.57	100	10.43	0.00	
Ensemble	NL	96.88	100	3.12	0.00	0.79
	AL	93.435	100	6.565	0.00	
	FL	88.76	100	11.24	0.00	

Then, in a rather non-trivial case, the tool has used its previous one-time training on cache-based SCAs to detect Spectre and Meltdown attacks. This is relatively difficult execution scenario as these two attacks are no bache-based SCAs and the tool has no a priori knowledge of their behavior. Intuitively, retraining ML models involving variants of Spectre and Meltdown attacks would increase the detection accuracy and further reduce the chances of detection inconsistencies (discussed in Section VI-B).

Tables 18 and 19 present our detection results on Meltdown and Spectre attacks, respectively, using the WHISPER tool. Although the performance of WHISPER is evaluated using the same evaluation metrics, in these cases, sampling is adjusted to a fixed interval of 100 ms rather than a percentage of attack completion. This is because both these attacks are independent of cryptosystems. Therefore, they do not have a single victim process. Moreover, the overhead mainly constitutes the sampling cost of HPCs and not of any victim's slowdown. With a rather fixed and coarse-grain sampling rate, the overhead for the detection module is much less than in previous cases. As shown in Table 17, detection accuracy of all individual ML models under NL and AL conditions remains very high and the Ensemble consequently also provides very high accuracy. In the FL condition, however, the detection accuracy suffers and the tool generates a large number of False Positives. The fact that WHISPER is able to achieve reasonably good detection accuracies under 2 out of 3 load conditions is because both Spectre and Meltdown attacks use CSCAs to extract information from caches, and their behavior thus resembles that of the conventional CSCAs, which WHISPER is able to capture with no further training. An important result in the case of Meltdown detection is the fact that there are no False Negatives in the system. Similar pattern in results can be observed in Table 19 for Spectre attack detection. In this case, the overhead is slightly higher than in the detection of Meltdown. Figures 13–16 illustrate the variations in previously selected HPCs under NL and FL for Spectre and Meltdown attacks, respectively.

TABLE 18: Results on the detection of Meltdown Attack using the WHISPER tool without any modifications

ML Model	Load	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	96.428	100	3.57	0.0	0.49
	AL	97.435	100	2.56	0.00	
	FL	42.857	100	57.14	0.00	
SVM	NL	92.857	100	7.14	0.00	0.70
	AL	97.435	100	2.56	0.00	
	FL	87.142	100	12.85	0.00	
RF	NL	96.428	100	3.57	0.00	0.47
	AL	97.435	100	2.56	0.00	
	FL	42.857	100	57.14	0.00	
Ensemble	NL	96.428	100	3.57	0.00	0.79
	AL	97.435	100	2.56	0.00	
	FL	42.857	100	57.14	0.00	

TABLE 19: Results on the detection of Spectre Attack using the WHISPER tool without any modifications

ML Model	Load	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	97.058	100	2.94	0.00	1.14
	AL	91.176	100	8.82	0.00	
	FL	42.857	100	57.14	0.00	
SVM	NL	100.0	100	0.00	0.00	0.99
	AL	94.117	100	5.88	0.00	
	FL	87.142	100	12.85	0.00	
RF	NL	97.058	100	2.94	0.00	1.28
	AL	94.117	100	5.88	0.00	
	FL	42.857	100	57.14	0.00	
Ensemble	NL	97.058	100	2.94	0.00	1.30
	AL	94.117	100	5.88	0.00	
	FL	42.857	100	57.14	0.00	

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

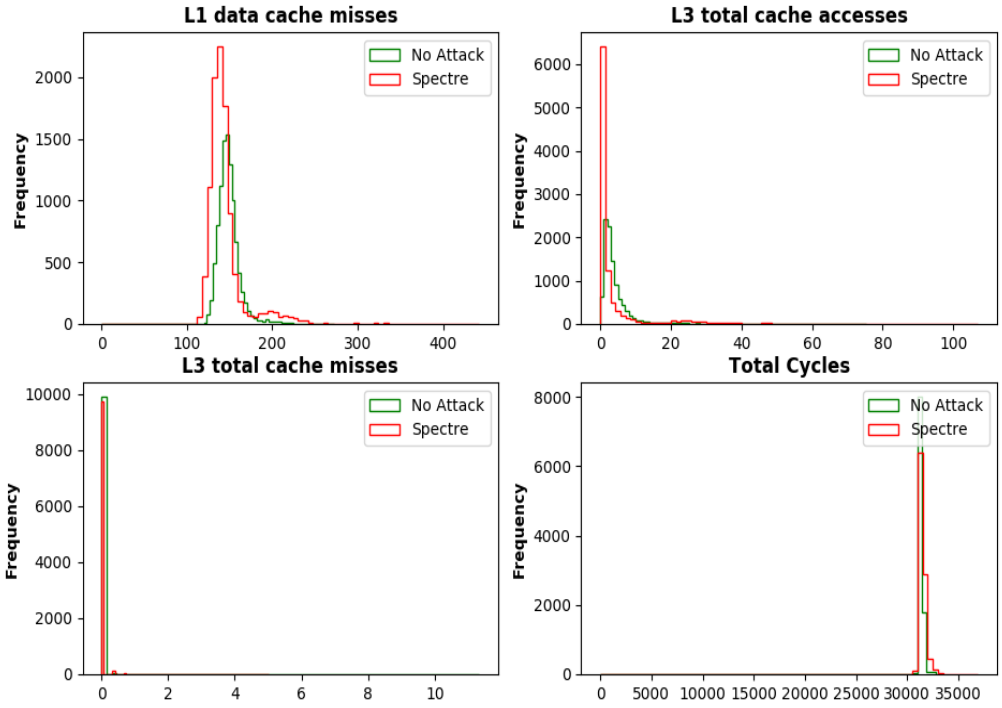


FIGURE 13: Run-time behavior of selected hardware events under No Load conditions for Spectre Attack

The significant variations in detection accuracy, particularly in the FL condition, originates from the fact that WHISPER did not use any additional training on the data set generated for Meltdown and Spectre attacks. Rather, it uses its previous training on Flush+Flush, Flush+Reload and Prime+Probe attacks to also detect Spectre and Meltdown, which become unknown attacks for WHISPER in this case.

B. SCENARIO 02: WITH MODIFICATION IN THE WHISPER TOOL

Section VI-A described results obtained with no modifications to the WHISPER tool. Intuitively, if the data set relevant to Spectre and Meltdown attacks is used in the training phase, the high rate of FPs in the previous case would naturally be reduced. Moreover, the use of new features that would make it possible to capture the behavior of these attacks more specifically can further enhance accuracy. Thus, we demonstrate that if WHISPER is enhanced with more specific features and retrained, its reported accuracy gets even better.

As mentioned earlier, the baseline vulnerability exploited by a Spectre attack is the mistraining of branch instructions, i.e., the branch direction predictor or BTB. Therefore, we select two new hardware events related to the Spectre attack: (1) Total Branch Instructions, and (2) Total Branch Mispredictions, as features for the ML models. Our experiments and analysis of Spectre attack code reveals that it has the much less number of total instructions to execute and, among those instructions, a large proportion of instructions comprise of branch instructions. Interestingly enough, the attack process generates the highest number of branch mispredictions. Our results show that, if total branch instructions and the total branch mispredictions generated by a process are collected as features, the ML models can detect Spectre attack with very high accuracy, as shown in Table 20 under all load conditions.

In the case of a Meltdown attack, similar observations can be made by observing the number of page faults generated by the attacker. Meltdown generates segmentation faults or invalid page faults while bypassing the memory isolation through exceptions. Page faults can be captured through a software event by using Perf API. Our experiments reveal that, for a relatively very small number of total instructions, the Meltdown attack process generates significantly large numbers of page faults that can serve as a very effective and highly uncorrelated feature for ML models in WHISPER. When we used page faults as one of the features, the detection accuracies against the Meltdown attack were significantly enhanced, as shown in Table 21.

VII. COMPARATIVE ANALYSIS AND DISCUSSION OF THE RESULTS–LESSONS LEARNED

Table 22 provides a brief comparative analysis of WHISPER with the state-of-the-art. We provide this comparison with respect to the evaluation metrics used in this work, *i.e.*, detection accuracy, speed, performance overhead and system load conditions in order to provide the reader with an overview of the existing techniques. Details on these techniques are already discussed in Section II. The authors in [59] detected Flush+Reload at an accuracy of F-score=0.93 and speed of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

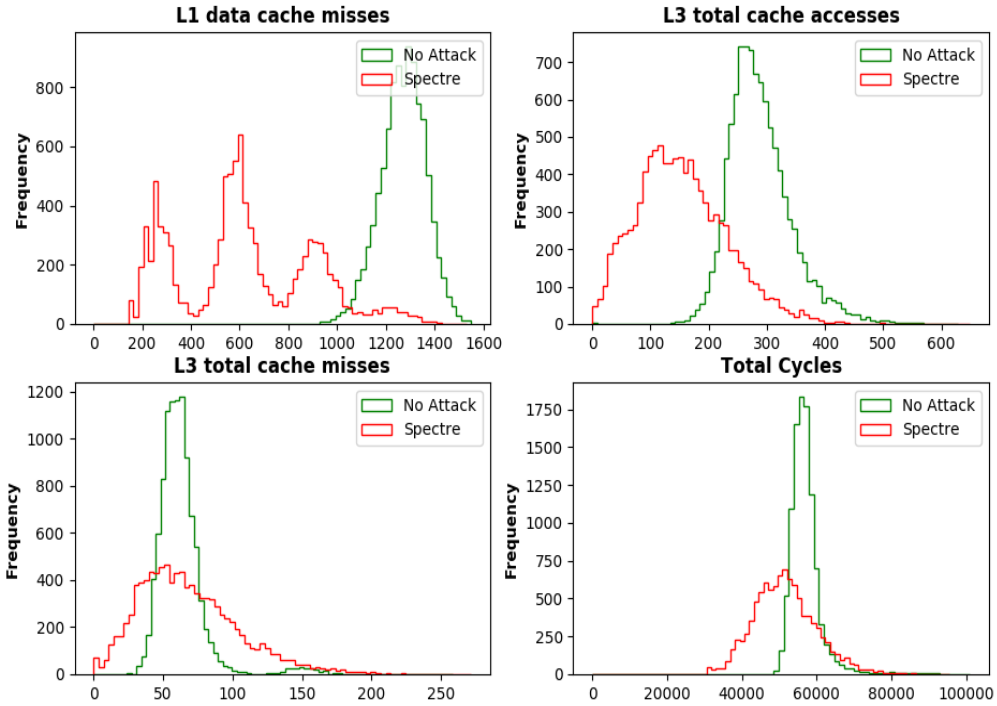


FIGURE 14: Run-time behavior of selected hardware events under Full Load conditions for Spectre Attack

TABLE 20: Results on the detection of Spectre attack using the WHISPER tool with modified features and retraining of ML Models.

Model	Loads	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	99.93	100	0.07	0	1.6
	AL	99.06	100	0.57	0.37	
	FL	98.03	100	1.18	0.79	
RF	NL	99.97	100	0.03	0	1.6
	AL	98.40	100	1.27	0.33	
	FL	97.36	100	1.98	0.66	
SVM	NL	99.25	100	0.69	0.06	1.6
	AL	97.29	100	2.02	0.69	
	FL	95.87	100	2.87	1.26	
Ensemble	NL	99.80	100	0.17	0.03	1.6
	AL	99.13	100	0.57	0.29	
	FL	97.43	100	1.56	1.01	

TABLE 21: Results on the detection of Meltdown attack using the WHISPER tool with modified features and retraining of ML Models.

Model	Loads	Accuracy (%)	Speed (ms)	FP (%)	FN (%)	Overhead (%)
DT	NL	99.95	100	0.05	0	2.11
	AL	99.83	100	0.13	0.04	
	FL	98.27	100	1.24	0.49	
RF	NL	99.35	100	0.65	0	2.11
	AL	97.39	100	1.98	0.63	
	FL	94.67	100	3.43	1.90	
SVM	NL	99.97	100	0.03	0	2.11
	AL	99.17	100	0.67	0.16	
	FL	98.24	100	1.39	0.37	
Ensemble	NL	99.43	100	0.48	0.09	2.11
	AL	99.17	100	0.55	0.28	
	FL	98.69	100	0.79	0.52	

the 1/5th of attack completion in the presence of background noise. The F-score is a measure of accuracy in statistical analysis of binary classification. However, this technique does not discuss the impact on performance degradation.*HexPADS* [34] claims to have detected Flush+Reload and Prime+Probe attacks with 100% accuracy with 2% overhead. These results are reported under No load conditions, which may lead to erronous accuracy and increased overhead under noisy system load conditions. There is no discussion of how fast *HexPADS* can detect any of the tested attacks. *CloudRadar* [24] claims to have detected Prime+Probe and Flush+Reload attacks with 100% accuracy and 5% overhead under No Load conditions as well. This technique also suffers from the same issues as *HexPADS*. The techniques proposed in [15] detect Flush+Reload and Prime+Probe attacks at 97% and 98% accuracy, respectively, and within 2% detection speed in the presence of background noise but did not discuss the over-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

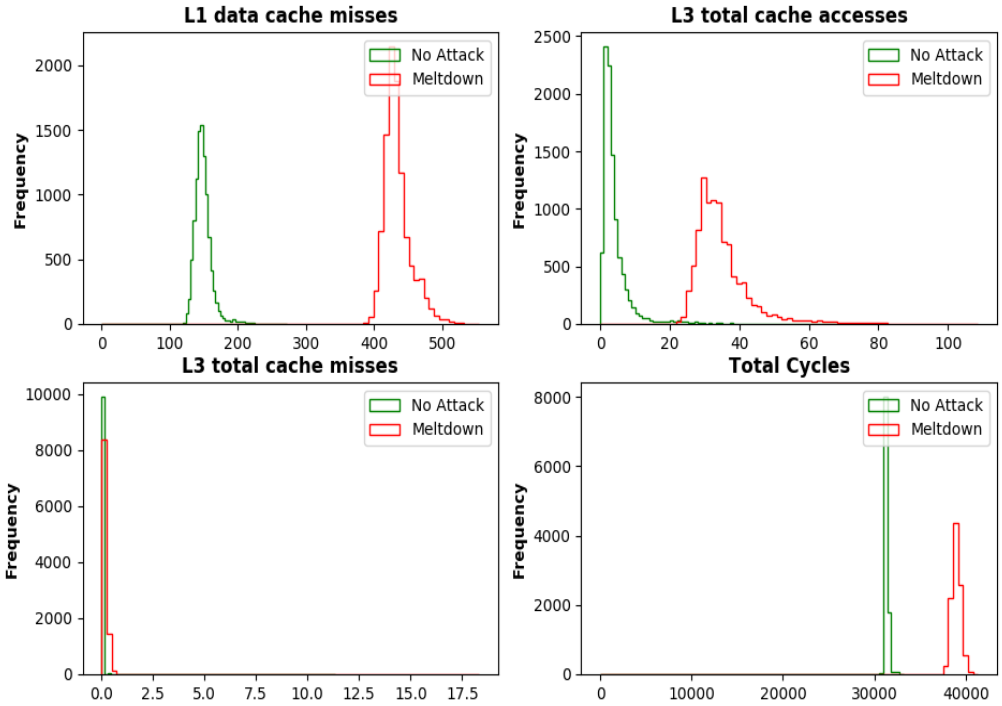


FIGURE 15: Run-time behavior of selected hardware events under No Load conditions for Meltdown Attack

head of their proposed mechanism. The technique proposed in [60] claims to detect template attacks with 99% accuracy but does not provide detection speed, overhead and any variations in system load conditions. The technique proposed in [35] claims to work 100% accurately on Prime+Probe attack with 2% overhead. However, the authors do not provide any result on the detection speed and the percentage of false negatives increases under load conditions with the proposed technique. The solution proposed in *SCADET* [33] claims to detect Flush+Reload at 100% accuracy in isolated conditions but there is no discussion about the impact of detection speed and overhead under load conditions. Similarly, The *SCADET* technique proposed in [33] claims to perform 100% accurately under different load conditions but the detection speed and the performance overhead of this mechanism are not discussed, on the other hand, it raises false alarms in load conditions and trace timing is long in some cases, which is not suitable for early stage detection.

The brief comparison provided in Table 22 supports our earlier discussion that detection accuracy alone is not a suit-able measure of a good detection technique. Based on these comparisons, our proposed detection tool clearly performs better under stringent evaluation metrics. Our mechanism works for a larger set of attacks while performing runtime detection. We provide results for multiple attacks, namely: Flush+Reload on AES, Flush+Flush on AES half-key (FF_Imp1), Flush+Flush on AES full-key (FF_Imp2), Prime+Probe on AES half-key (FF_Imp1), Prime+Probe on AES full-key (FF_Imp2), Spectre and Meltdown. The reported accuracy of our tool for these attacks is comparable or better than the state-of-the-art techniques. Similarly, the reported detection speed is well within the theoretical upper bound of 50% attack completion and performance overhead remains low under variable load conditions.

A. DISCUSSION

Throughout this work, we have built our argument in favor of detection-based protection, which would help apply mitigation only after successful detection of CSCAs. Our experiments applied to three different case studies, comprising the state-of-the-art CSCAs, demonstrate that detection can be highly accurate with a minimum system overhead at runtime and fast enough to raise the alarm before attack completion. Our experiments with different attack categories enabled us to provide an evidence-based analysis of CSCA detection.

The first lesson we learned from these results is that almost all CSCAs, whether they are known or unknown, dependent or in-dependent of cryptosystem, leave their footprints on the cache hierarchy, either in the form of access timing or access pattern. Such a footprint can be captured by carefully profiling the behavior of the affected process(es) without a priori knowledge of the type of attack or the temporal order of multiple attacks taking place. To this end, selection of most relevant hardware events is of paramount importance, as demonstrated in Section IV-B1. Nevertheless, it is pertinent to mention here that the underlying hardware events may be imprecise, non-deterministic and limited in number, which can lead to an increased error rate (FPs & FNs) under smarter attacks in the future. Das *et al.* [65] provide a detailed insight into the limitations and pitfalls of using HPCs for security.

We have provided the proof of concept on the use of WHISPER for a larger attack vector. For instance, we have shown detection results for cache-based attacks like Flush+Reload, Flush+Flush and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

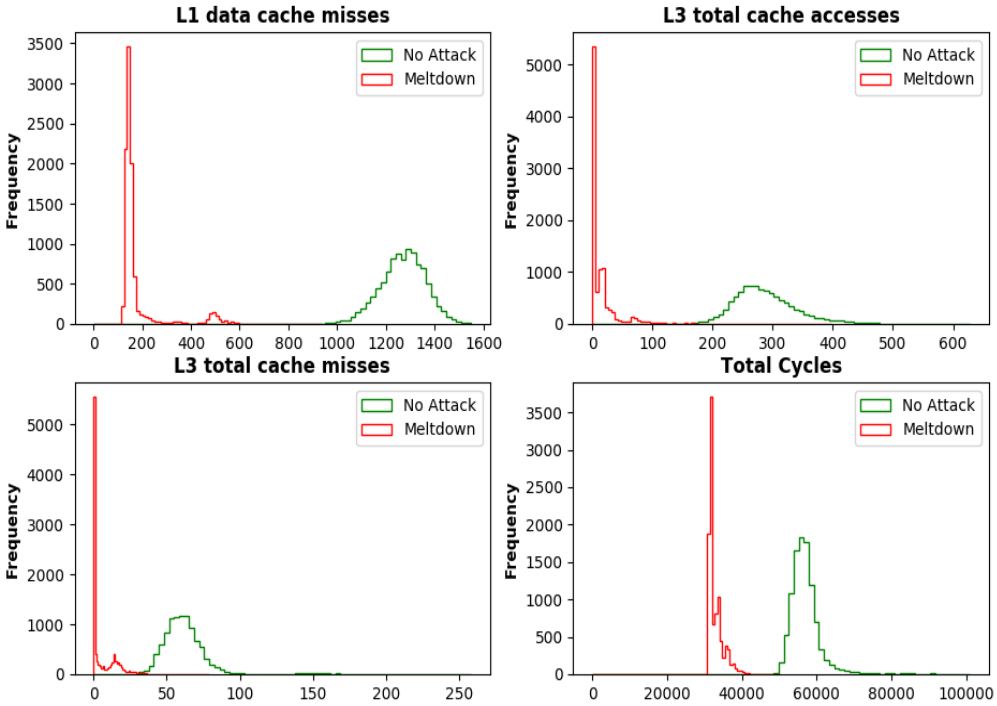


FIGURE 16: Run-time behavior of selected hardware events under Full Load conditions for Meltdown Attack

Prime+Probe as well as for computational attacks like Spectre Meltdown. We show that WHISPER is able to detect available state-of-the-art attacks that target cryptosystems as well as those that are independent of cryptosystems. Our results show significant high detection accuracy for these attacks with reasonably low performance overhead, making WHISPER a very good candidate for adoption for early-stage runtime detection.

The second lesson we learned from our experiments is that simple statistical or threshold-based solutions are not sufficient to distinguish anomalous behavior from normal behavior, particularly in the case of side-channel attacks. As shown in Section IV-C, the attacks can take place in any temporal order and the data being collected by the hardware events might not be easy to classify. In certain cases, even stand-alone ML models might not be sufficient to detect anomalous behavior, as illustrated in our third case study with The SVM model (Section V-D3). Our experiments with 12 different ML models and the use of Ensemble model provide empirical evidence to strengthen the belief that machine learning can help building the resilient software/hardware security solutions for modern computing systems. We have demonstrated their success on known CSCAs.

The third lesson we learned is that detection tools and techniques must be evaluated in a holistic manner, *i.e.*, by considering security as well as performance aspects. For instance, very high detection accuracy is not enough if the tool cannot be adopted for runtime detection due to its slow speed or considerably high runtime overhead, which would cause significant slowdown for the victim's process. The use of multiple stringent evaluation metrics in this work reveals that there is a trade-off between the performance overhead and the detection speed of a runtime detection tool. In order to serve as the first line of defense against SCAs, a detection tool must be fast enough to report an attack before its completion and yet it has to be light-weight enough to continuously monitor the system's behavior without significantly increasing the overhead. Our experiments show that increased detection granularity yields more reliable results in terms of speed, accuracy and misclassification rates, resulting in increased performance overhead. With decreased granularity, performance overhead is significantly reduced. Therefore, we propose to use the WHISPER tool in two different modes, *i.e.*, fine-grain and coarse-grain sampling modes. The tool offers this flexibility to run in either of these modes depending on the operating conditions and persisting threat levels. Once a threat is detected, the tool can adjust its sampling rate to confirm the detection and subsequently report to the OS to take remedial measures.

The fourth and last lesson we learned is that, unless there are design changes at the hardware level and a complete overhauling of the entire computing stack, software alone cannot completely protect against side-channel information leakage. It can only make such leakage harder. Detection solutions that are entirely based on software are vulnerable to adversarial attacks, which could corrupt software features for ML models in order to overcome the detection mechanism. From an in-depth analysis and experimentation in this work, we have learned that it is hard to corrupt the values of hardware events directly collected using HPCs at runtime. In order to do so, the malicious processes need to alter the behavior of caches at the hardware level to camouflage their activities, which is much harder than tempering software-based values. Therefore, hypothetically, even if the adversary knows the detection mechanism as a white box, it is still very hard to manipulate the information of hardware events at runtime.

VIII. CONCLUSIONS AND FUTURE WORK

This paper argued in favor of using runtime detection as the first line of defense against cache side-channel attacks. We advocate for a detection-based protection mechanisms as existing mitigation techniques against SCAs either completely remove or greatly reduce the performance benefits of resource sharing. In this paper, we propose a machine learning based CSCA detection tool, called WHISPER, for Intel $x86$ architecture. The tool comprises multiple machine learning models, integrated in an Ensemble fashion, that use real-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

TABLE 22: Comparative analysis of WHISPER with the state-of-the-art [Note: F+R=Flush+Reload, F+F=Flush+Flush, P+P=Prime+Probe, NA=Not Available].

Ref.	Attack	Accuracy	Speed	Overhead	Load
[59]	F+R	F-score 0.93	1/5th of attack completion	NA	Yes (Apache)
[34]	F+R, P+P	100%	NA	<2%	No
[24]	P+P, F+R	100%	NA	<5%	No
[15]	F+R, P+P	97%, 98%	2%	NA	Yes (SPEC)
[32]	Template Attacks	>99%	NA	NA	No
[60]	P+P	100%	NA	2%	Yes (CIW)
[35]	F+R	100%	NA	NA	No
[33]	P+P	100%	NA	NA	Yes (Open source)
This work	F+R(AES), F+F(AES, Imp1), F+F(AES, Imp2), P+P(AES, Impl1), P+P(AES, Imp2), Spectre, Meltdown	99.99%, 99.99%, 99.8% , 99.99%, 99.77%, 97.05%, 97.43%	<40%, <25%, <25%, <0.21%, <0.21%, fixed, fixed	< 8%	Yes (SPEC)

time behavioral data of concurrent processes running on Intel $x86$ architecture. The WHISPER tool is capable of detecting a large set of the state-of-the-art attacks without the need to retrain its machine learning models for each specific type of attack. We describe extensive experimentation with six different attacks and evaluate the tool under stringent constraints, such as: detection accuracy, speed, performance overhead and distribution of error (*i.e.*, false positives and false negatives). Our results show very high detection accuracy, *i.e.*, $> 99\%$, with a negligible error rate. We have also demonstrated the scalability of WHISPER by detecting computational attacks, *i.e.*, Spectre and Meltdown. Results show that WHISPER, without retraining and no modification of the hardware events, is able to detect both vulnerabilities with a reasonable accuracy. The tool is light weight and easy to integrate for runtime detection. We provide experimental evaluation of the tool under variable load conditions to demonstrate its resilience and consistency in noisy environment. As a future direction, specialized machine learning models can be trained to detect partially known or fully unknown attacks.

REFERENCES

[1] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, "Computer security: principles and practice". Pearson Education Upper Saddle River, NJ, USA, 2012.

[2] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: a timing attack on OpenSSL constant-time RSA," Journal of Cryptographic Engineering, vol. 7, no. 2, pp. 99–112, 2017.

[3] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in 23rd USENIX Conference on Security Symposium, 2014.

[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache Attack," in Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721, DIMVA 2016, 2016.

[5] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES". Springer Berlin Heidelberg, 2006.

[6] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-VM attack on AES," in International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 299–319, 2014.

[7] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," Journal of Cryptographic Engineering, pp. 1–27, 2016.

[8] X. Jin, H. Chen, X. Wang, Z. Wang, X. Wen, Y. Luo, and X. Li, "A simple cache partitioning approach in a virtualized environment," in IEEE Int'l Symposium on Parallel & Distributed Processing with Applications, pp. 519–524, Aug 2009.

[9] F. Liu and R. B. Lee, "Random Fill Cache Architecture," in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 203–215, 2014.

[10] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud," in USENIX Security, pp. 189–204, 2012.

[11] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," Applied Soft Computing, vol. 49, pp. 1162–1174, 2016.

[12] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "NIGHTs-WATCH: A Cache-based Side-channel Intrusion Detector Using Hardware Performance Counters," in Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP '18, (New York, NY, USA), pp. 1:1–1:8, ACM, 2018.

[13] M. Mushtaq, A. Akram, M. Bhatti, N. B. R. Rao, V. Lapotre, and G. Gogniat, "Run-time detection of Prime+Probe side-channel attack on AES encryption algorithm," in Global Information Infrastructure and Networking Symposium, Greece, 2018.

[14] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V. Lapotre, and G. Gogniat, "Machine Learning For Security: The Case of Side-Channel Attack Detection at Run-time," in 25th IEEE International Conference on Electronics Circuits and Systems, Bordeaux, FRANCE, 2018.

[15] Z. Allaf, M. Adda, and A. Gegov, "A comparison study on Flush+Reload and Prime+Probe attacks on AES using machine learning approaches," UK Workshop on Computational Intelligence, 2017.

[16] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," Journal of Hardware and Systems Security, vol. 2, no. 1, pp. 33–50, 2018.

[17] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in International Conference on Cryptographic Hardware and Embedded Systems (CHES), vol. 9813, pp. 368–388, 08 2016.

[18] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in Proc. of 24th USENIX Conf. on Security Symp., (Berkeley, CA, USA), pp. 897–912, USENIX Assoc., 2015.

[19] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "FORESHADOW: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018., pp. 991–1008, 2018.

[20] D. Genkin, L. Valenta, and Y. Yarom, "May the Fourth Be With You: A microarchitectural side channel attack on several real-world applications of Curve25519," in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pp. 845–858, 2017.

[21] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

attacks: Exploiting speculative execution," in 40th IEEE Symposium on Security and Privacy (S&P'19), 2019.
[22] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in 27th USENIX Security Symposium (USENIX Security 18), 2018.
[23] M. Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment," IEEE Transactions on Cloud Computing, vol. 2, pp. 395–408, Oct 2014.
[24] T. Zhang, Y. Zhang, and R. B. Lee, "CloudRadar: A real-time side-channel attack detection system in clouds," in International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, 2016.
[25] F. Liu, H. Wu, K. Mai, and R. B. Lee, "Newcache: secure cache architecture thwarting cache side channel attacks," IEEE Micro Special Issues on Security, vol. 36, 2016.
[26] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," SIGARCH Comput. Archit. News, vol. 35, pp. 494–505, June 2007.
[27] W.-M. Hu, "Reducing timing channels with fuzzy time," Journal of computer security, vol. 1, no. 3-4, pp. 233–254, 1992.
[28] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in ACM CCS, (NY, USA), 2012.
[29] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-VM side-channels," in 23rd USENIX Security Symposium, (San Diego, CA), 2014.
[30] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in 28th USENIX Security Symposium (USENIX Security 19), pp. 249–266, 2019.
[31] G. Irazoqui, K. Cong, X. Guo, H. Khattri, A. K. Kanuparthi, T. Eisenbarth, and B. Sunar, "Did we learn from LLC side channel attacks? A cache leakage detection tool for crypto libraries," CoRR, vol. abs/1709.01552, 2017.
[32] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks." Cryptology ePrint Archive, Report 2017/564, 2017. https://eprint.iacr.org/2017/564.
[33] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: a side-channel attack detection tool for tracking Prime+Probe," in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018.
[34] M. Payer, HexPADS: A Platform to Detect "Stealth" Attacks, pp. 138–154. Cham: Springer International Publishing, 2016.
[35] S.-h. PENG, Q.-f. ZHOU, and J.-l. ZHAO, "Detection of cache-based side channel attack based on performance counters," DEStech Transactions on Computer Science and Engineering, no. aiie, 2017.
[36] A. Raj and J. Dharanipragada, "Keep the PokerFace on! Thwarting cache side channel attacks by memory bus monitoring and cache obfuscation," Journal of Cloud Computing, vol. 6, no. 1, p. 28, 2017.
[37] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: cross-cores cache covert channel," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 46–64, Springer, 2015.
[38] A. Barresi, K. Razavi, M. Payer, and T. R. Gross, "CAIN: Silently Breaking ASLR in the Cloud," in 9th USENIX Workshop on Offensive Technologies (WOOT), 2015.
[39] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," Black Hat, vol. 15, p. 71, 2015.
[40] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with déjá vu," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 7–18, ACM, 2017.
[41] S. Briongos, G. Irazoqui, P. Malagón, and T. Eisenbarth, "CacheShield: Detecting cache attacks through self-observation," in Proceedings of the 8th Conference on Data & Application Security & Privacy, pp. 224–235, ACM, 2018.
[42] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, "SpyDetector: An approach for detecting side-channel attacks at runtime," IJIS, 2018.
[43] "Online repository of cache side-channel attacks," https://github.com/ECLab-ITU, 2019.
[44] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP '15, (Washington, DC, USA), pp. 605–622, IEEE Computer Society, 2015.
[45] Nepoche, "https://github.com/nepoche/flush-reload," 2017.
[46] D. Gruss, "https://github.com/iaik/flush_flush," 2017.
[47] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient Cache Attacks on AES, and Countermeasures," Journal of Cryptology, vol. 23, no. 1, pp. 37–71, 2010.
[48] C. Percival, "Cache missing for fun and profit," in Proc. of BSDCan 2005, 2005.
[49] O. Aciicmez, B. B. Brumley, and P. Grabher, "New results on instruction cache attacks," in Proc. of Int'l Conference on Cryptographic Hardware and Embedded Systems, CHES'10, (Heidelberg), pp. 110–124, Springer-Verlag, 2010.
[50] O. Aciicmez and W. Schindler, "A vulnerability in RSA implementations due to instruction cache analysis and its demonstration on OpenSSL," in Proceedings of the 2008 The Cryptopgraphers' Track at the RSA Conference on Topics in Cryptology, CT-RSA'08, (Berlin, Heidelberg), pp. 256–273, Springer-Verlag, 2008.
[51] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in Xen," in CCSW, 2011.
[52] "Intel 64 and ia-32 arch. software developer's manual volume 3b: System programming guide, part2.," June 2014.
[53] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics). Berlin, Heidelberg: Springer-Verlag, 2006.
[54] PerfMon, "$https://knowledge.ni.com/$," 2018.
[55] OProfile, "$http://oprofile.sourceforge.net/$," 2018.
[56] P. Tool, "$http://lacasa.uah.edu/$," 2018.
[57] I. V-Tune, "$https://software.intel.com/en-us/vtune-amplifier-cookbook$," 2018.
[58] "Performance application programming interface," in $http://icl.cs.utk.edu/papi/$, 2018.
[59] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," Journal of Applied Soft Computing, vol. 49, pp. 1162–1174, Dec. 2016.
[60] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in Fog and Mobile Edge Computing (FMEC), 2018 Third International Conference on, pp. 7–12, IEEE, 2018.
[61] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," IEEE Transactions on Neural Networks, vol. 20, no. 3, pp. 542–542, 2009.
[62] G. James, D. Witten, T. Hastie, and R. Tibshirani, An introduction to statistical learning, vol. 112. Springer, 2013.
[63] J. Friedman, T. Hastie, and R. Tibshirani, The elements of statistical learning, vol. 1. Springer series in statistics New York, 2001.
[64] J. Depoix and P. Altmeyer, "Detecting spectre attacks by identifying cache side-channelattacks using machine learning," in WAMOS 2018 Fourth Wiesbaden Workshop on Advanced Microkernel Operating Systems, Wiesbaden, Germany., Hochschule RainMan, Computer Engineering Department, August 2018.
[65] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, & perils of using hardware performance counters for security," IEEE Symposium on Security & Privacy, 2019.

MARIA MUSHTAQ Maria Mushtaq is a CNRS Post Doctoral researcher at LIRMM, University of Montpellier (UM), France. She did her Ph.D. from Lab-STICC, University of South Brittany (UBS), France in 2019. Maria has specific expertise in developing runtime detection and mitigation solutions against side-channel information leakage in computing systems. Her research interests mainly focus on cryptanalysis, constructing and validating software security components, and constructing OS-based security primitives against various hardware vulnerabilities.

JEREMY BRICQ Jeremy Bricq is a PhD student from the Université Catholique de Louvain, Belgium, since 2019. He spent 3 months internship in the Université de Bretagne-Sud, France, where he worked on the Side-Channel Attacks detection. Jeremy received a Master in Cybersecurity from the Université Libre de Bruxelles. His current works focus on Cryptography, with Fully Homomorphic Encryption.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.2988370, IEEE Access

MUHAMMAD KHURRAM BHATTI is an Assistant Professor at Information Technology University, Lahore, Pakistan. He did his Postdoc from KTH Royal Institute of Technology, Stockholm, Sweden. He did his MS and Ph.D. in Embedded Systems from University of Nice-Sophia Antipolis, France. His research interests include Embedded Systems, Information Security at both hardware and software level, Cryptanalysis, Mixed Criticality and Parallel Computing Systems.

AYAZ AKRAM Ayaz Akram received his bachelor's degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, and his master's degree in computer engineering from Western Michigan University, USA. He is currently pursuing a Ph.D. degree in computer science with the University of California at Davis, USA. His research interests include computer architecture, high-performance computing, and the intersection of computer architecture with other areas like computer security and machine learning.

VIANNEY LAPOTRE Vianney LAPOTRE received his MSc and PhD in Electrical and Computer Engineering from the University Bretagne-Sud, France. He spent six months as an invited researcher at the Ruhr-University of Bochum, Germany. He was a Postdoctoral at LIRMM, Montpellier, France. He is an associate professor at University Bretagne-Sud, France. His research interests include hardware security, reconfigurable and self-adaptive multiprocessor architectures.

GUY GOGNIAT is a Professor in ECE with the University of Bretagne-Sud, Lorient, France, where he has been since 1998. In 2005, he spent one year as an invited Researcher with the University of Massachusetts, Amherst, USA, where he worked on embedded security using reconfigurable technologies. His work focuses on embedded systems design methodologies and tools. He also conducts research in the domain of reconfigurable and adaptive computing and embedded system security.

PASCAL BENOIT received the Ph.D. degree in microelectronics from the University of Montpellier, France, in 2004, and the Habilitation degree from the Montpellier Laboratory of Informatics, Robotics and Microelectronics, University of Montpellier, in 2015. He was a Scientific Assistant with the Karlsruhe Institute of Technology, University of Karlsruhe, Germany. Since 2005, he has been a Permanent Associate Professor with the Montpellier Laboratory of Informatics, Robotics and Microelectronics, University of Montpellier. He has co-authored over 130 publications in books, journals, and conference proceedings. He holds five patents. His research interests include the Internet of Things, from smart sensors to gateways, energy efficiency, and security issues.

11 Conception conjointe d'un processeur et d'une chaîne de compilation pour lutter contre les attaques par canaux auxiliaires exploitant les variations temporelles durant l'exécution de code

Les travaux présentés dans cette section sont actuellement réalisés dans le cadre du projet *SCRATCHS* (Side-Channel Resistant Applications Through Co-designed Hardware/Software), financé par le labEx CominLabs, qui a débuté en 2021 pour une durée de 3 ans. Ce projet rassemble des compétences en conception d'architectures matérielles, en cybersécurité et en méthode formelle. Bien qu'au moment de l'écriture de ce manuscrit l'étude et le développement des premières contributions soient en cours, ce projet illustre la direction que je souhaite donner à l'axe de recherche "protections face aux attaques exploitant la microarchitecture".

11.1 Le projet SCRATCHS

Ce projet a pour but de co-concevoir un processeur implémentant le jeux d'instructions RISC-V [10] et sa chaîne de compilation afin de garantir par construction la protection d'applications sensibles face à des attaques par canaux auxiliaires exploitant des variations temporelles à l'exécution. En particulier, le projet se focalise sur les attaques exploitant la hiérarchie mémoire. La figure 21 illustre l'approche proposée ainsi que le rôle des partenaires du projet.

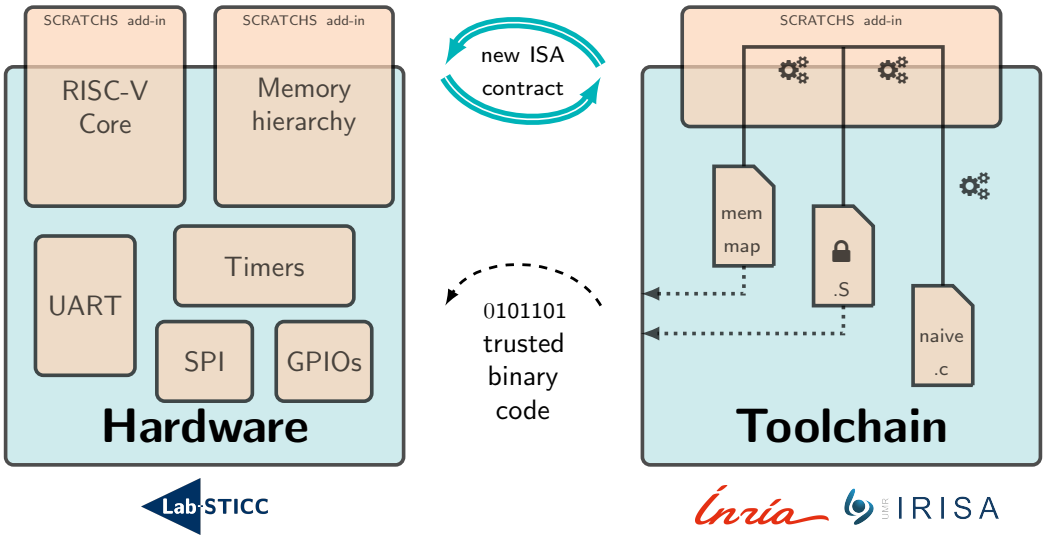


FIGURE 21 – Vue d'ensemble du projet SCRATCHS

Dans le but de permettre l'exécution sécurisée et performante d'applications sensibles, une approche populaire est de proposer une exécution du programme sensible en temps constant [Barthe 2020] [Cauligi 2019] [Wu 2018]. Cependant, cela ne peut être atteint via des solutions purement logicielles ou purement matérielles. En effet, même si le binaire généré par le compilateur peut être considéré temps constant après analyse, cette propriété peut être mise à mal par une méconnaissance du comportement réel du processeur au niveau microarchitectural (hiérarchie mémoire, prédiction, spéculation, ...). De plus, ces solutions pénalisent généralement fortement les performances temporelles de l'application. Les contre-mesures matérielles existantes se concentrant sur la protection de la hiérarchie mémoire se basent généralement sur le partitionnement

10. https://riscv.org/technical/specifications/

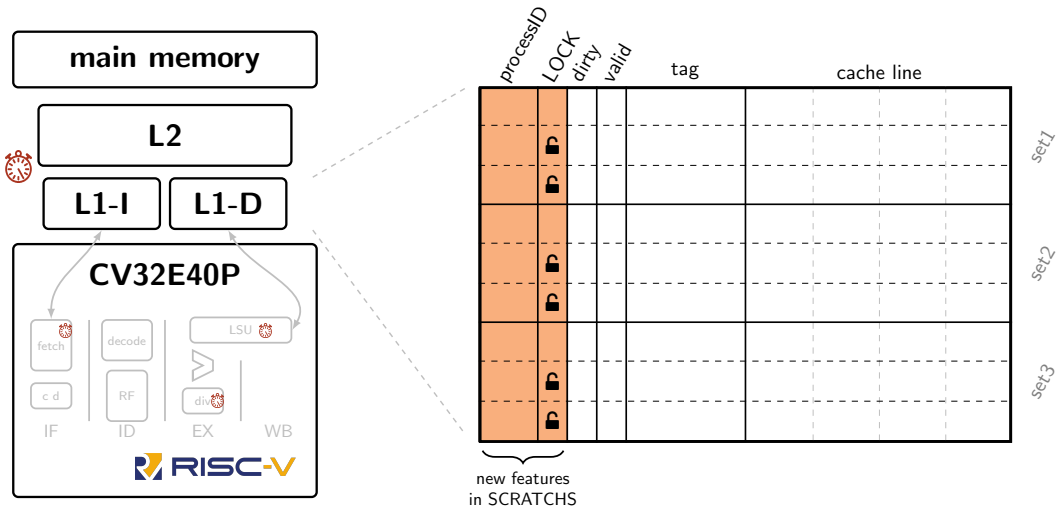


FIGURE 22 – Proposition d'extension de la mémoire cache

(i.e. [Wang 2007]), le verrouillage (i.e. [Mittal 2016]), ou l'ajout d'aléa dans les mémoires caches (i.e. [Werner 2019]). Ces différentes approches ont été principalement construites afin de fournir des protections génériques qui ne prennent pas en compte les exigences des applications. Cela conduit à des mécanismes rigides et coûteux.

Pour aller au-delà de ces limitations, le projet SCRATCHS plaide en faveur d'une collaboration plus étroite entre le logiciel et le matériel afin de proposer une protection prenant en compte les besoins applicatifs pour offrir le niveau de sécurité nécessaire tout en maintenant une utilisation efficace de la hiérarchie mémoire.

11.2 Hypothèses et modèle de menace

Le projet SCRATCHS se focalise sur des systèmes embarqués à très faible coût exécutant un système d'exploitation léger qui ordonnance un ensemble d'applications dont au moins une manipule des données sensibles dont la confidentialité doit être protégée.

Nous considérons un attaquant capable d'exécuter un code malveillant sur la plateforme via une mise à jour logicielle ou l'installation d'un logiciel tiers par l'utilisateur. De plus, l'application malveillante est capable de réaliser des mesures de temps précises via l'utilisation de compteurs de performances matériels. Enfin, nous considérons que l'attaquant possède une connaissance fine des codes sources de l'application victime. Cette dernière hypothèse est réaliste car l'application victime peut entièrement ou grandement reposer sur des librairies dont les sources sont publiquement disponibles.

11.3 Premières réflexions

Dans le cadre de ce projet de recherche, via la thèse de M. Nicolas Gaudin (2021-2024) que je co-encadre, le laboratoire Lab-STICC est en charge de la partie matérielle du projet. L'objectif est de fournir un système sur puce basé sur un processeur RISC-V intégrant les extensions proposées durant le projet. Deux premières extensions sont actuellement en cours de développement. La première, consiste à ajouter un mode de fonctionnement *temps constant* au processeur dont l'activation est réalisée par le logiciel (e.g. via l'écriture d'un registre de configuration). Ce mode permet de forcer la réalisation de certaines opérations (e.g. la division) en temps constant. La seconde, illustrée en figure 22 consiste à étendre les mémoires caches pour permettre l'utilisation, par le logiciel, d'instructions nommées *LOCK* et *UNLOCK* permettant le verrouillage de certaines lignes de cache.

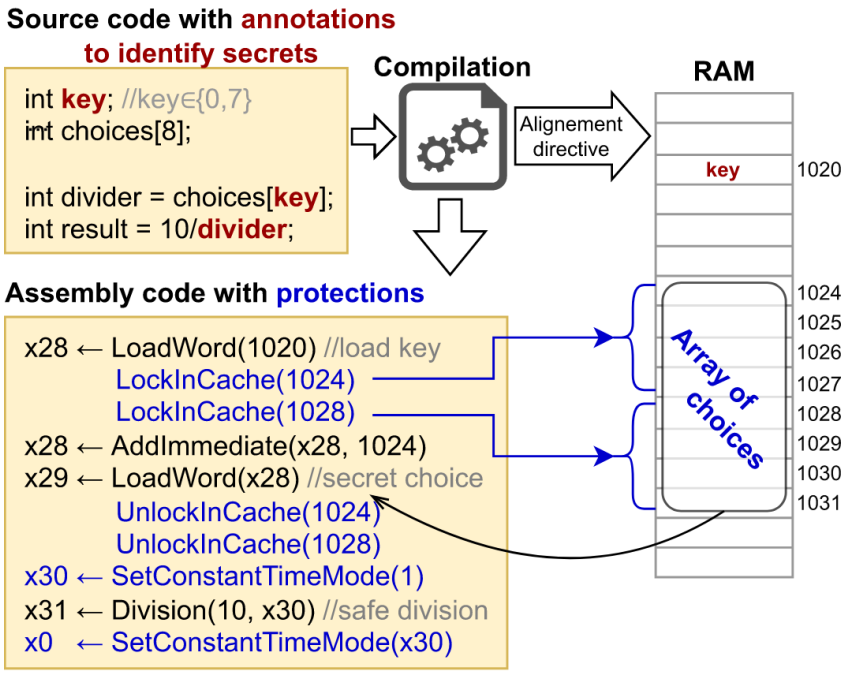


FIGURE 23 – Vue d'ensemble de l'utilisation par le logiciel des extensions matérielles proposées

L'utilisation de ces deux extensions au niveau logiciel est présentée dans la figure 23. Dans cet exemple, le code source est annoté pour identifier les variables sensibles à protéger. La chaîne de compilation est ensuite responsable de la génération d'un code intégrant l'utilisation des mécanismes de protections proposés dans le projet. Le code assembleur présenté dans la figure 23 illustre l'utilisation des deux extensions matérielles précédemment présentées pour 1) protéger un accès mémoire dépendant de la variable sensible *key* en verrouillant préalablement le tableau *choices* en mémoire cache avant son utilisation et 2) en passant le processeur dans un mode temps constant avant de réaliser une opération de division sensible dont le temps d'exécution aurait pu renseigner l'attaquant sur la valeur du diviseur.

Ce projet étant actuellement en cours, nous espérons publier nos premières contributions durant l'année 2023. Les mécanismes présentés ci-dessus seront complétés durant le projet pour tenir compte de différentes évolutions au niveau matériel (TLB, prédiction de branchement, hiérarchie mémoire complexifiée, plateforme multi-coeur, ...) et logiciel (prise en charge par le système d'exploitation, optimisations utilisant l'analyse statique, ...).

12 Conclusion

Les attaques exploitant la microarchitecture des processeurs modernes ont montré ces dernières années qu'aucune architecture de processeur n'est actuellement capable de garantir un haut niveau de sécurité. Les architectes ont passé des décennies à concevoir des mécanismes permettant d'optimiser les performances temporelles, qui aujourd'hui, ouvrent la porte à de nombreuses menaces. Dans ce contexte, les travaux de recherche que j'ai conduis ont permis d'aborder la problématique sous deux angles différents.

De 2014 à 2019, les travaux menés ont permis de proposer des solutions palliatives sans modification de l'architecture matérielle. Les solutions proposées s'appuient sur des briques logicielles pour détecter une attaque ou pour isoler les applications sensibles en s'appuyant sur des ressources matérielles dédiées. Cependant, ces solutions sont temporaires et de nouvelles architectures de processeurs doivent être proposées en intégrant, dès la conception, des objectifs en terme de sécurité. C'est dans ce contexte, que depuis 2021, les travaux de recherche que je mène propose de revoir la conception matérielle des processeurs embarqués dans le but de fournir des solutions matérielles de confiance. Le projet SCRATCH, actuellement en cours, s'inscrit parfaitement dans cette ligne directrice en s'appuyant sur une co-conception du logiciel et du matériel dans le but de fournir une solution offrant des garanties de sécurité élevées et une maîtrise des performances.

La table 13 fournie quelques éléments de bilan des activités de cet axe de recherche.

Table 13 – bilan des activités pour l'axe de recherche "Protections face aux attaques exploitant la microarchitecture"

Valorisation	6 revues et 8 communications internationales 2 communication nationale
Collaborateurs	G. Gogniat (UBS), K. Bhatti (ITU), D. Goehringer (TU Dresden), P. Cotret (ENSTA Bretagne), G. Hiet et P. Wilke (Centrale Supelec), F. Besson (INRIA)
Doctorants	M. Mendez-Real, M. Mushtaq, N. Gaudin
Financeurs	ANR, LabEx CominLabs, Université Bretagne Sud, Région Bretagne

12.1 Liste des publications associées à l'axe

12.1.1 Articles dans des revues internationales avec comité de lecture

[MUS 2022] M. Mushtaq, M. M. Yousaf, M. K. Bhatti, V. Lapotre, G. Gogniat, **The Kingsguard OS-level mitigation against cache side-channel attacks using runtime detection**, Annals of Telecommunications, Springer, 2022.

[MUK 2020a] MM. A. Mukhtar, M. Mushtaq, M. K. Bhatti, V. Lapotre, G. Gogniat,**FLUSH + PREFETCH : A Countermeasure Against Access-driven Cache-based Side-Channel Attacks**, Journal of Systems Architecture, vol 104, 2020.

[MUS 2020a] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, P. Benoit, **WHISPER A Tool for Run-time Detection of Side-Channel Attacks**, IEEE Access, vol 8, pp. 83871-83900, 2020.

[AKR 2020a] A. Akram, M. Mushtaq, M. K. Bhatti, V. Lapotre, G. Gogniat, **Meet the Sherlock Holmes' od Side Channel Leakage : A Survey of Cache SCA Detection Techniques**,

IEEE Access, vol 8, pp. 70836-70860, 2020.

[MUS 2020b] M. Mushtaq, M. A. Mukhtar, V. Lapotre, M. K. Bhatti, G. Gogniat, **Winter is here ! A decade of cache-based side-channel attacks, detection & mitigation for RSA**, Information Systems, vol 92, 2020.

[MEN 2018a] M. Mendez Réal, P. Wehner, V. Lapotre, D. Göhringer and G. Gogniat, **Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators**, in ACM Transactions on Embedded Computing Systems (TECS), vol. 12, n. 2, Article 55, February 2018.

12.1.2 Communications internationales avec comité de lecture et actes

[MUS 2019a] M Mushtaq, A Akram, M.K. Bhatti, U. Ali, V Lapotre, G Gogniat, **Sherlock Holmes of Cache Side-Channel Attacks in Intel's x86 Architecture**, in Proc. of IEEE Conference on Communications and Network Security (CNS), 2019.

[MUS 2018a] M Mushtaq, A Akram, M.K. Bhatti, M. Chaudhry, V Lapotre, G Gogniat, **NIGHTs-WATCH : a cache-based side-channel intrusion detector using hardware performance counters**, in Proc. of 7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP), 2018.

[MUS 2018b] M Mushtaq, A Akram, M.K. Bhatti, R.N. Bin Rais, V Lapotre, G Gogniat, **Run-time Detection of Prime+Probe Side-Channel Attack on AES Encryption Algorithm**, in Proc. of Global Information Infrastructure and Networking Symposium (GIIS), 2018.

[MUS 2018c] M Mushtaq, A Akram, M.K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V Lapotre, G Gogniat, **Machine Learning For Security : The Case of Side-Channel Attack Detection at Run-time**, in Proc. of 25th IEEE International Conference on Electronics (ICECS), 2018.

[MEN 2016a] M. Méndez Real, V. Migliore, V. Lapotre and G. Gogniat, **ALMOS many-core operating system extension with new secure-aware mechanisms for dynamic creation of secure zones**, in Proc. of 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2016.

[MEN 2016b] M. Mendez Réal, V. Migliore, V. Lapotre, G. Gogniat, P. Wehner, and D. Göhringer, **Dynamic Spatially Isolated Secure Zones for NoC-based Many-core Accelerators**, in Proc. of 8th IEEE International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2016.

[MEN 2016c] M. Mendez Réal, V. Migliore, V. Lapotre, G. Gogniat, P. Wehner, J. Rettkowski and D. Göhringer, **MPSoCSim extension : An OVP Simulator for the Evaluation of Cluster-based Multicore and Many-core architectures**, 4rd Workshop on Virtual Prototyping of Parallel and Embedded Systems (ViPES) as part of the International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS), 2016.

12.1.3 Communications internationales invitées sans actes

[MUS 2018d] M. Mushtaq, A. Akram, M. Bhatti, V. Lapotre, G. Gogniat, **Cache-Based Side-Channel Intrusion Detection using Hardware Performance Counters**, 16th Internatio-

nal Workshops on Cryptographic architectures embedded in logic devices, Lorient, France, 2018.

12.1.4 Communication nationale avec comité de lecture et actes

[MEN 2016d] M. Méndez Real, V. Lapotre, G. Gogniat and V. Migliore, **Physical isolation against cache-based Side-Channel Attacks on NoC-based architectures**, Compas'2016.

12.1.5 Communications nationales invitées sans actes

[LAP 2018a] V. Lapotre, **Déploiement sécurisé d'applications au sein des architectures many-cœurs**, FETCH 2018.

12.1.6 Communication par affiche

[DEV 2015a] C. Dévigne, J-B. Bréjon, Q. Meunier, F. Wajsbürt, M. Mendez Real, V. Migliore, V. Lapotre, G. Gogniat, M. Aichouch, M. Ait Hmid, C. Mancillas López, L. Bossuet, V. Fischer, **Applications security in manycore platform, from operating system to hypervisor : how to build a chain of trust**, CHES 2015 Workshop on Cryptographic Hardware and Embedded Systems, 2015.

References

[Aciiçmez 2007a] Onur Aciiçmez, Çetin Kaya Koç et Jean-Pierre Seifert. *On the power of simple branch prediction analysis.* In Proceedings of the 2nd ACM symposium on Information, computer and communications security, pages 312–320, 2007.

[Acıiçmez 2007b] Onur Acıiçmez, Çetin Kaya Koç et Jean-Pierre Seifert. *Predicting secret keys via branch prediction.* In Cryptographers' Track at the RSA Conference, pages 225–242. Springer, 2007.

[Aciicmez 2007c] Onur Aciicmez et Jean-Pierre Seifert. *Cheap hardware parallelism implies cheap security.* In Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), pages 80–91. IEEE, 2007.

[Almaless 2014] Ghassan Almaless. *Operating System Design and Implementation for Single-Chip cc-NUMA Many-Core.* PhD thesis, Université Pierre et Marie Curie, 2014.

[Barthe 2020] Gilles Barthe, Sandrine Blazy, Benjamin Grégoire, Rémi Hutin, Vincent Laporte, David Pichardie et Alix Trieu. *Formal verification of a constant-time preserving C compiler.* Proc. ACM Program. Lang., vol. 4, no. POPL, pages 7 :1–7 :30, 2020.

[Cauligi 2019] Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala et Deian Stefan. *FaCT : a DSL for timing-sensitive computation.* In Kathryn S. McKinley et Kathleen Fisher, editeurs, Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019, pages 174–189. ACM, 2019.

[Ge 2018] Qian Ge, Yuval Yarom, David Cock et Gernot Heiser. *A survey of microarchitectural timing attacks and countermeasures on contemporary hardware.* Journal of Cryptographic Engineering, vol. 8, no. 1, pages 1–27, 2018.

[Gras 2018] Ben Gras, Kaveh Razavi, Herbert Bos et Cristiano Giuffrida. *Translation leak-aside buffer : Defeating cache side-channel protections with TLB attacks.* In 27th USENIX Security Symposium (USENIX Security 18), pages 955–972, 2018.

[Gruss 2016] Daniel Gruss, Clémentine Maurice, Klaus Wagner et Stefan Mangard. *Flush+Flush : a fast and stealthy cache attack.* In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 279–299. Springer, 2016.

[Gullasch 2011] David Gullasch, Endre Bangerter et Stephan Krenn. *Cache games–bringing access-based cache attacks on AES to practice.* In 2011 IEEE Symposium on Security and Privacy, pages 490–505. IEEE, 2011.

[Hund 2013] Ralf Hund, Carsten Willems et Thorsten Holz. *Practical timing side channel attacks against kernel space ASLR.* In 2013 IEEE Symposium on Security and Privacy, pages 191–205. IEEE, 2013.

[Kocher 2019] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz et Yuval Yarom. *Spectre Attacks : Exploiting Speculative Execution.* In 40th IEEE Symposium on Security and Privacy (S&P'19), 2019.

[Lipp 2018] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom et Mike Hamburg. *Meltdown : Reading Kernel Memory from User Space.* In 27th USENIX Security Symposium (USENIX Security 18), 2018.

[Lou 2021] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang et Yinqian Zhang. *A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography.* ACM Computing Surveys (CSUR), vol. 54, no. 6, pages 1–37, 2021.

[Méndez Real 2016] Maria Méndez Real, Vincent Migliore, Vianney Lapotre, Guy Gogniat, Philipp Wehner, Jens Rettkowski et Diana Göhringer. *MPSoCSim extension : An OVP Simulator for the Evaluation of Cluster-based Multicore and Many-core architectures.* In 4rd Workshop on Virtual Prototyping of Parallel and Embedded Systems (ViPES) as part of the International Conference on Embedded Computer Systems : Architectures, Modeling, and Simulation (SAMOS), Samos, Greece, Juillet 2016.

[Mittal 2016] Sparsh Mittal. *A Survey of Techniques for Cache Locking.* ACM Trans. Des. Autom. Electron. Syst., vol. 21, no. 3, Mai 2016.

[Mushtaq 2020a] Maria Mushtaq, Jeremy Bricq, Muhammad Khurram Bhatti, Ayaz Akram, Vianney Lapotre, Guy Gogniat et Pascal Benoit. *WHISPER : A Tool for Run-Time Detection of Side-Channel Attacks.* IEEE Access, vol. 8, pages 83871–83900, 2020.

[Mushtaq 2020b] Maria Mushtaq, Muhammad Asim Mukhtar, Vianney Lapotre, Muhammad Khurram Bhatti et Guy Gogniat. *Winter is here ! A decade of cache-based side-channel attacks, detection & mitigation for RSA.* Information Systems, vol. 92, page 101524, 2020.

[Mushtaq 2022] Maria Mushtaq, Muhammad Muneeb Yousaf, Muhammad Khurram Bhatti, Vianney Lapotre et Gogniat Guy. *The Kingsguard OS-level mitigation against cache side-channel attacks using runtime detection.* Annals of Telecommunications - annales des télécommunications, Janvier 2022.

[Osvik 2006] Dag Arne Osvik, Adi Shamir et Eran Tromer. *Cache attacks and countermeasures : the case of AES.* In Cryptographers' track at the RSA conference, pages 1–20. Springer, 2006.

[Percival 2005] Colin Percival. *Cache missing for fun and profit*, 2005.

[Real 2018] Maria Méndez Real, Philipp Wehner, Vianney Lapotre, Diana Göhringer et Guy Gogniat. *Application Deployment Strategies for Spatial Isolation on Many-Core Accelerators.* ACM Trans. Embed. Comput. Syst., vol. 17, no. 2, feb 2018.

[Tromer 2010] Eran Tromer, Dag Arne Osvik et Adi Shamir. *Efficient cache attacks on AES, and countermeasures.* Journal of Cryptology, vol. 23, no. 1, pages 37–71, 2010.

[Wang 2006] Zhenghong Wang et Ruby B Lee. *Covert and side channels due to processor architecture.* In 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), pages 473–482. IEEE, 2006.

[Wang 2007] Zhenghong Wang et Ruby B. Lee. *New Cache Designs for Thwarting Software Cache-based Side Channel Attacks.* SIGARCH Comput. Archit. News, vol. 35, no. 2, pages 494–505, Juin 2007.

[Werner 2019] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss et Stefan Mangard. *ScatterCache : Thwarting Cache Attacks via Cache Set Randomization.* In 28th USENIX Security Symposium (USENIX Security 19), pages 675–692, Santa Clara, CA, Ao ?t 2019. USENIX Association.

[Wu 2018] Meng Wu, Shengjian Guo, Patrick Schaumont et Chao Wang. *Eliminating timing side-channel leaks using program repair.* In Frank Tip et Eric Bodden, editeurs, ISSTA, pages 15–26. ACM, 2018.

[Yarom 2014] Yuval Yarom et Katrina Falkner. *FLUSH+RELOAD : A High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In 23rd USENIX security symposium (USENIX security 14), pages 719–732, 2014.

Quatrième partie

Contributions à l'axe : sécurité des processeurs embarqués

Dans cette partie, je présente les travaux de recherches menés dans le cadre de l'étude, la conception et l'implémentation de protections contre les attaques logiques et physiques au sein de processeurs embarqués. L'objectif de cet axe est de proposer des solutions s'appuyant sur des éléments matériels afin de minimiser le coût lié à la sécurité en terme de performance et d'énergie.

Dans une première section, une courte introduction à la problématique est proposée. Puis, une section est dédiée à la description des différents travaux réalisés dans le cadre de cet axe de recherche. Pour chacune de ces sections, je donne une brève introduction au sujet d'étude puis, je décris les principales contributions et enfin je fournis une sélection des principaux articles illustrant les travaux réalisés.

Les travaux décrits dans cet axe correspondent à un projet collaboratif et deux projets de thèse de doctorat que je co-encadre actuellement :

— Co-conception logicielle/matérielle pour le Dynamic Information Flow Tracking (2015 - 2019)

— Rejeu matériel d'instructions face aux attaques par injection de fautes (2019 - 2023)

— Protection d'un processeur avec DIFT contre des attaques physiques (2021 - en cours)

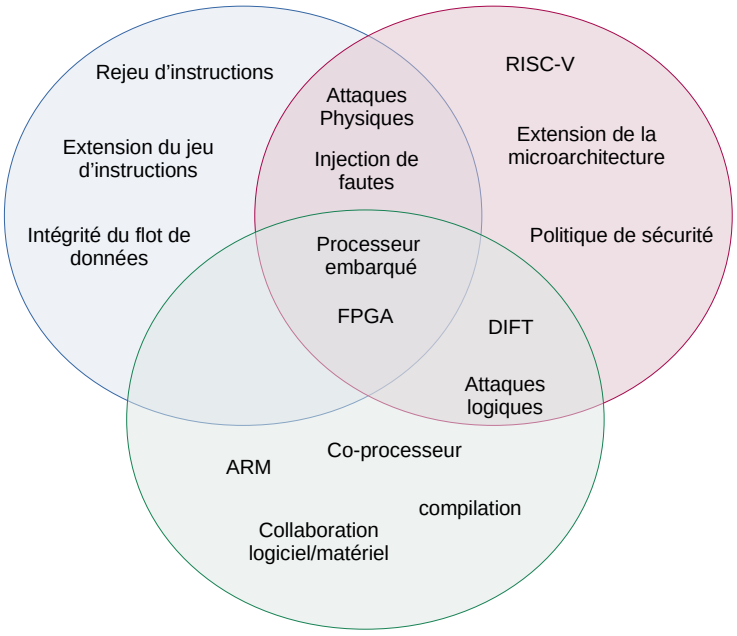


FIGURE 24 – Couverture thématique de l'axe sécurité des processeurs embarqués

La figure 24 présente des mots clés associés à chacun de ces travaux de recherche. Cette représentation permet de mettre en évidence les liens entre ces trois projets. Les différents travaux se focalisent principalement sur la conception de processeurs embarqués ciblant une implantation sur FPGA. Parmi ces projets, deux s'intéressent à la problématique de la protection face aux attaques logiques à travers des mécanismes de *Dynamic Information Flow Tracking* (DIFT). Le premier de ces projets, qui s'est déroulé entre 2015 et 2019, s'est intéressé au développement d'un co-processeur matériel dédié pour permettre l'accélération des traitements nécessaires au DIFT pour un processeur ARM. Ce projet propose une solution originale mêlant analyse statique et

instrumentation du programme à protéger, extraction d'information à l'exécution via les composants de débogage du processeur et traitements DIFT sur FPGA. Le second projet, débuté en 2021, couvre une autre dimension du problème en s'intéressant à l'impact des attaques physiques sur la robustesse de ce type de mécanisme. En particulier, nous nous intéressons ici à des attaques par injection de fautes ciblant un mécanisme DIFT *in-core* intégré à la microarchitecture d'un processeur RISC-V.

Le dernier projet, débuté en 2019 et dont la thèse associée a été soutenue en janvier 2023, se focalise sur l'étude et le développement de protections matérielles face à des attaques physiques par injection de fautes. Ces travaux se concentrent sur la protection d'un processeur embarqué très faible coût à travers l'intégration au processeur d'un mécanisme de rejeu d'instructions piloté par le logiciel via une extension du jeu d'instructions.

13 Co-conception logicielle / matérielle pour le Dynamic Information Flow Tracking

Les travaux présentés dans cette section ont été réalisés entre 2015 et 2019 dans le cadre du projet collaboratif HardBlare financé par le LabEx CominLabs. L'objectif de ce projet consistait à repousser les limites des approches DIFT (Dynamic Information Flow Tracking) purement logicielles en proposant une solution reposant sur la co-conception logicielle/matérielle dans laquelle un ensemble de traitements coûteux sont déportés sur un co-processeur matériel dédié.

13.1 DIFT matériel

Les techniques de DIFT [Brant 2021] permettent de détecter diverses attaques logicielles en attachant et en propageant, à l'exécution, des (*tags*) à des conteneurs d'information (fichiers, registres, pages mémoires, etc.). Associées à une politique de gestion de flux d'information, elles permettent de détecter des violations de la politique de sécurité instanciée par le programmeur (par exemple, accès à une zone mémoire non autorisée, transmission de données non autorisées).

Les solutions de DIFT peuvent permettre une analyse au sein d'un système d'exploitation [Efstathopoulos 2005][Zeldovich 2008][Roy 2009][Georget 2017] via l'analyse des appels systèmes. Cependant, cette approche ne peut que sur-approximer le comportement des applications, menant à de nombreux faux positifs. De plus, elle ne permet pas de détecter des attaques de plus bas niveau (p.ex. un détournement du flot de contrôle) requérant une analyse DIFT au niveau des registres processeurs. Pour palier à cette limitation, des techniques de DIFT ont été proposées au niveau application [Qin 2006][Newsome 2005] pour analyser chaque instruction exécutée. Cependant, ces analyses sont intégrées dans le code de l'application, engendrant généralement des surcôuts importants et une absence d'isolation entre la solution de sécurité et l'application protégée. Les traitements liés au DIFT peuvent alors être déportés sur des ressources matérielles dédiées.

Trois grandes approches permettent de réaliser ce déport[Kannan 2009]. Ces approches sont appelées *In-core DIFT* [Palmiero 2018][Dhawan 2015][Dalton 2007][Suh 2004], *Offloading DIFT* [Chen 2008][Ruwase 2008] ou *Off-core DIFT* [Kannan 2009][Venkataramani 2008]. Dans l'approche *In-core DIFT*, la microarchitecture du processeur est étendue afin d'intégrer les traitements nécessaires dans chaque étage du pipeline. Par conséquent, la lecture, la vérification et la propagation des tags sont réalisées en parallèle de l'exécution des instructions du programme exécuté. Pour cela, le banc de registre et la hiérarchie mémoire sont étendus afin de stocker les tags associés aux données manipulées. Dans l'approche *Offloading DIFT*, les traitements sont déportés, au sein d'un système multi-coeur, sur un coeur dédié. Une zone mémoire partagée entre les deux coeurs est utilisée pour échanger les informations nécessaires au fonctionnement du DIFT. Cette approche à l'avantage de ne pas nécessiter de modifications matérielles du processeur mais requiert l'utilisation d'un coeur supplémentaire dédié au mécanisme de protection. Enfin, dans l'approche *Off-core DIFT* un coprocesseur dédié au traitement DIFT est utilisé. Comparé à l'approche précédente, cela permet de préserver des ressources puisque que ce coprocesseur est bien plus simple et optimisé qu'un coeur générique. Cette approche nécessite toutefois des modifications du coeur dans le but de transmettre au coprocesseur la trace d'exécution de l'application à protéger. Cette limitation est adressée par le projet HardBlare dont l'approche est présentée dans la section suivante.

13.2 Approche proposée

Le projet Harblare s'appuie sur une co-conception logicielle / matérielle afin de proposer un mécanisme de DIFT ciblant des systèmes embarqués complexes intégrant un système d'exploitation (e.g Linux) et exécutant des applications sensibles nécessitant un haut niveau de sécurité. Bien que la solution proposée puisse être qualifiée d'approche *Off-core DIFT*, elle ne nécessite

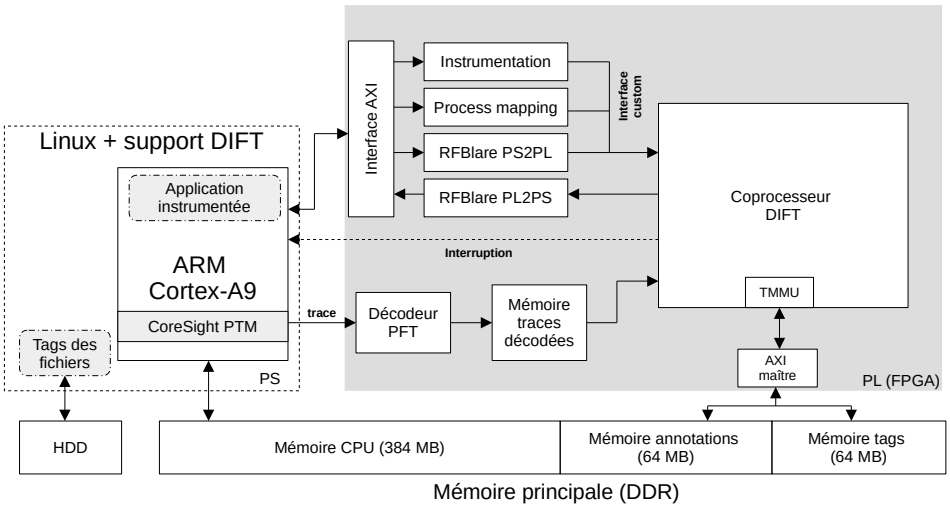


FIGURE 25 – Vue d'ensemble de l'approche proposée dans le cadre du projet HardBlare

pas de modification du processeur. Le défis majeur consiste alors à mettre en oeuvre des moyens alternatifs permettant d'extraire les informations décrivant le comportement du code exécuté nécessaires aux analyses DIFT. Dans le cadre de ce projet, ces informations sont rassemblées via trois canaux : des annotations générées à la compilation, une instrumentation du code et l'utilisation des informations de debogage générées par des composants dédiés du processeur cible. Dans le cas de ce projet, le composant ARM CoreSight Program Trace Macrocell (TPM) du processeur ARM Cortex-A9 est considéré.

La figure 25 illustre l'approche proposée. Les travaux auxquels j'ai participé, via le co-encadrement de M. Arnab kumar Biswas et ma participation à l'encadrement de thèse de M. Muhammad Abdul Wahab, se concentrent sur le module matériel du projet (nommée *PL (FPGA)* dans la figure 25). Ce module agrège des données de plusieurs sources pour réaliser ses traitements : 1) les annotations, générées à la compilation et stockées dans une zone mémoire dédiée nommée *Mémoire tags*, décrivant les propagations de tags à réaliser par le coprocessesseur lors de l'exécution d'un bloc d'instructions par le processeur ; 2) les tags associés au contenu de la mémoire principale stockés dans une zone mémoire dédiée nommée *Mémoire annotations* ; 3) les informations issues de l'instrumentation du programme permettant au co-processeur de prendre connaissance des adresses de données en mémoire non connues à la compilation (obtenues via le module *Instrumentation*) et 4) la trace d'exécution décodée (*décodeur PFT*) du programme exécuté via le composant de debogage ARM CoreSight PTM du processeur (*CS components*). Ces données sont alors traitées par le coprocesseur DIFT en parallèle de l'exécution du programme sur le processeur afin de déterminer si une violation de la politique de sécurité est advenue. Dans l'affirmative, un signal d'interruption est levé et le programme en cours d'exécution est suspendu. Le composant *RFBlare PS2PL* (resp. *RFBlare PL2PS*) est utilisé lors de la lecture (resp. l'écriture) d'un fichier. Lors de la lecture, le tag associé au fichier géré par le système d'exploitation est transmis au coprocesseur. Lors de l'écriture, le tag généré par le coprocesseur est transmis au système d'information pour être stocké comme métadonnée du fichier.

Le coeur de la solution matérielle est le coprocesseur DIFT dont la figure 26 présente le détail. Ce co-processeur se compose de deux coeurs distincts. Le premier, nommé *Dispatcher* et basé sur une architecture de processeur MIPS, est en charge de traiter les traces extraites du composant de debogage ARM CoreSight PTM et décodées par le *PFT decoder*. À partir des informations des traces, le logiciel exécuté sur ce coeur permet d'identifier le bloc d'instructions en cours d'exécution, lire en mémoire les annotations correspondantes et transmettre ces dernières au second coeur du coprocesseur DIFT. Le second coeur, nommé *Tag Management Core* (TCM), exécute les annotations transmises par le *Dispatcher* afin de réaliser la propagation des tags et la

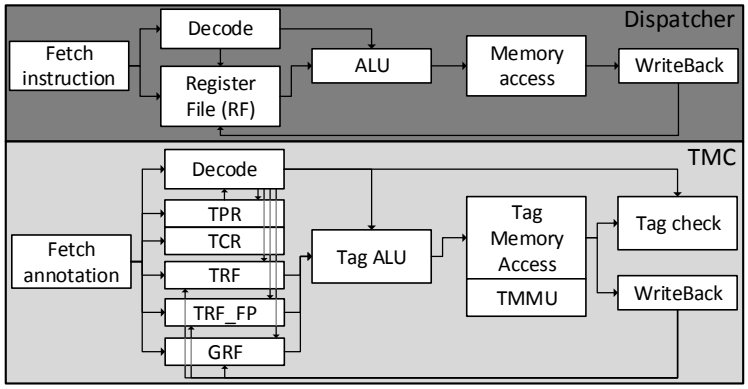


FIGURE 26 – Architecture interne du coprocesseur DIFT

vérification du respect de la politique de sécurité. Les opérations à réaliser lors de la propagation des tags sont configurées via le registre TPR *Tag propagation Register*. La politique de sécurité est configurée via le registre TCR *Tag Check Register*. Ces deux registres sont configurés par le *Dispatcher* lors d'une phase d'initialisation du co-processeur. Il est important de noter que dans l'approche proposée dans le projet HardBlare, un TMC est dédié à la surveillance d'une unique application et d'une unique politique de sécurité. Il est toutefois possible d'instancier plusieurs TMC travaillant en parallèle pour surveiller plusieurs applications ou mettre en oeuvre différentes politiques de sécurité dans le but de détecter différents types d'attaques.

13.3 Principaux résultats

Le projet HardBlare a permis de construire un démonstrateur complet sur la base des travaux de thèse de M. Mounir Nasr Allah [Nasr Allah 2020] (partie logicielle du projet) et M. Muhammad Abdul Wahab [Wahab 2018a] (partie matérielle du projet). La solution a été implémentée et évaluée sur une plateforme Xilinx ZYNQ-7000 [11]. Pour la partie logicielle du projet, des versions étendues du noyau Linux, de la chaîne de compilation LLVM, de la librairie *C* ***musl*** et du *loader* ***ld.so*** on été proposées. Ces éléments ainsi que que les développements matériels sont disponibles publiquement [12].

L'impact sur les performances temporelles des applications protégées a été évalué à l'aide de la suite *MiBench benchmark* [Guthaus 2001]. Les résultats ont montré un surcoût important entre 4 et 12 fois le temps d'exécution original. Ces résultats sont principalement dû à l'instrumentation du binaire de l'application protégée. Cependant, ces coût devraient pouvoir être réduits en réalisant des analyses supplémentaires (p. ex une analyse statique de propagation des constantes permettant de prédire la valeur d'une adresse pointée par un registre).

En terme de sécurité, l'approche a été testée en mettant en oeuvre trois programmes vulnérables ciblant un dépassement de tampon sur la pile (CWE-121), l'intégrité des données (CWE-1214) et la fuite de données sensibles (CWE-200). Dans les trois scénarios, les attaques ont été correctement détectées.

La table 14 présente l'évaluation en surface des composants matériels développés durant le projet. La majorité des ressource du FPGA sont utilisées pour l'implémentation de l'interface AXI (6,75%), du *Dispatcher* (4,18%) et du TMC (3,45%). La solution complète utilise 20,4% des ressources disponibles. Cela laisse aisément la possibilité d'intégrer des composants TMC supplémentaires pour protéger plusieurs applications ou gérer plusieurs politiques de sécurité en parallèle. En effet, l'ajout d'un TMC et des composants matériels supplémentaires nécessaires représente 8% des ressources totales disponibles sur le FPGA cible (4095 slice LUTs, 9074 slice registers et 6 BRAM tiles). En considérant un taux d'utilisation des ressources disponibles de

11. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html
12. https://gitlab.inria.fr/blare/hardblare

TABLE 14 – Surface des composants matériels HardBlare

Nom du Composant	Slice LUTs (en %)	Slice Registers (en %)	BRAM Tile
Dispatcher	2223 (4.18%)	1867 (1.75%)	3
TMC	1837 (3.45%)	2581 (2.43%)	6
Décodeur PFT	121 (0.23%)	231 (0.22%)	0
Instrumentation	676 (1.27%)	2108 (1.98%)	0
RFBlare PS2PL	662 (1.24%)	2106 (1.98%)	0
RFBlare PL2PS	62 (0.12%)	56 (0.05 %)	0
Mémoire traces décodées	0	0	2
AXI maître	858 (1.61%)	2223 (2.09 %)	0
TMMU	295 (0.55%)	112(0.10 %)	3
Interface AXI	2733 (5.14%)	2495 (2.34 %)	0
Divers	1381 (2.6%)	2160 (2.03%)	0
Total Design	10848 (20.39%)	15939 (14.98%)	14 (10%)
Total FPGA	53 200	106 400	140

80%, il est possible de protéger 8 applications en parallèle en utilisant la plateforme Xilinx ZYNQ-7000.

Les contributions présentées dans cette section sont détaillées dans l'article ci-dessous publié en 2018 et présenté à la conférence *Reconfig 2018* [Wahab 2018b].

A small and adaptive coprocessor for information flow tracking in ARM SoCs

Muhammad Abdul Wahab [α], Pascal Cotret [β], Mounir Nasr Allah [δ], Guillaume Hiet [δ]
Arnab Kumar Biswas [γ], Vianney Lapôtre [γ], Guy Gogniat [γ]
[α] *IETR/SCEE research group, firstname.lastname@centralesupelec.fr*
[β] *Independent researcher, pascal.cotret@gmail.com*
[δ] *INRIA/CIDRE research group, firstname.lastname@centralesupelec.fr*
[γ] *Lab-STICC/University of South Brittany, firstname.lastname@univ-ubs.fr*

***Abstract*—DIFT (*Dynamic Information Flow Tracking*) has been a hot topic for more than a decade. Unfortunately, existing hardware DIFT approaches have not been widely used neither by research community nor by hardware vendors. It is due to two major reasons: current hardware DIFT solutions lack support for multi-threaded applications and implementations for hardcore processors. This work addresses both issues by introducing an approach with some unique features: DIFT for multi-threaded software, virtual memory protection (rather than physical memory as in related works) and Linux kernel support using an information flow monitor called RFBlare. These goals are accomplished by taking advantage of a notable feature of ARM CoreSight components (context ID) combined with a custom DIFT coprocessor and RFBlare. The communication time overhead, major source of slowdown in total DIFT time overhead, is divided by a factor 3.8 compared to existing solutions with similar software constraints as in this work. The area overhead of this work is lower than 1% and power overhead is 16.2% on a middle-class Xilinx Zynq SoC.**

1. Introduction

IFT (*Information Flow Tracking*) consists of adding tags to information containers, propagating and checking these tags during program execution. A tag is a value associated with the information container and represents a level of security. For instance, if only two tag values are considered, the tags can represent private or public information (a tag with N bits can represent 2^N security levels). When containers are processed in a program, tags must be updated: this step is known as *tags computation*. IFT can be used to detect or prevent software attacks such as buffer overflows, SQL injection or confidential data leakage [1]. There are two types of IFT:

- SIFT (*Static IFT*) consists of an off-line analysis of the binary to check that all branches of the program are trustworthy.
- DIFT (*Dynamic IFT* is performed at runtime: it monitors data flow of the program to make sure that no unauthorized operation is done in the current execution branch.

SIFT is an appealing solution since it does not introduce any runtime overhead and ensure that the program is safe before executing it. However, this approach is not suited to protect a whole system consisting of different complex applications developed using different languages. DIFT is a more practical and flexible solution that can take into account some information that is hard to predict before execution, such as dynamic memory allocation, data or code that depends on the program inputs, etc. In this article, we propose a hybrid approach. The DIFT monitor described in this work relies on annotations that were statically pre-computed at compile time.

First, DIFT has been implemented in software but the performance overhead of such solutions is their main drawback (the application runs 37 times slower than without DIFT [1]).

In the last decade, multiple architectures taking advantage of FPGAs have been proposed to reduce this performance overhead. This improvement comes at the expense of flexibility provided by software IFT. Moreover, there is currently no solution provided by CPU hardware vendors (for instance, ARM and Intel) that allows implementing DIFT. Solutions like ARM TrustZone or Intel SGX allow protecting memory regions and provide security features for trusted applications. However, no security guarantees are provided if applications are untrusted, which is the case in this work.

On the one hand, most of related works implement DIFT on softcores [1]–[4] allowing to test the approach quickly. But most of these architectures are not directly portable to hardcore CPUs. DIFT requires information at runtime, such as the current executed instruction or accessed memory addresses, which can be easily recovered from softcore CPUs through existing signals.

On the other hand, other works such as [5] rely on ARM-based SoCs which do not export enough information from the CPU. The common way of recovering information for DIFT on a hardcore CPU is through instrumentation. Instructions are added in the program in order to recover information that is used to compute and check tags. However, the performance overhead due to instrumentation is high: it can represent up to 90% of the total DIFT overhead [3]. Furthermore, the solution proposed in [3] is still based on a softcore CPU which is not the final target of this work.

Debug components can also be used to retrieve such information [5]. Depending on the type of debug components, the program may still need instrumentation to recover missing information. ARM CPUs include two debug components: ETM and PTM. If an ETM (*Embedded Trace Macrocell*) [6] with data trace component is included on the device, there is no need to instrument the code. The ETM sends an information for each CPU instruction executed. However, CPU that target performance-intensive systems using rich OS (*Operating System*), such as Cortex-A cores, use the PTM (*Program Trace Macrocell* [5]). This debug component, considered in this work, only sends information about branch or jump addresses. This solution requires to statically analyze and instrument the program to recover missing information in the PTM trace [5].

This work targets ARM hardcore CPUs running a Linux kernel. An MMU (*Memory Management Unit*) takes care of translating virtual addresses to physical addresses when executing applications. Existing works associate a tag to the physical address. However, Linux applications contain virtual addresses. To resolve the difference in these address spaces, related work [7] uses a lookup table to keep track of translations done by the kernel. In this work, we propose a novel solution to tag virtual memory instead of physical memory. It provides the advantage of avoiding recovery overhead of virtual to physical address translations.

Existing hardware-assisted DIFT solutions lack Linux kernel support. Most solutions do not target modern operating systems. Works considering Linux OS lack the information on how to initialize the tag of information containers and other kernel modifications. This is very important because if the initialization is not done correctly, DIFT will fail in detecting attacks. Therefore, all the kernel modifications we propose are explained in this

978-1-7281-1968-7/18/$31.00 ©2018 European Union

TABLE 1: Comparison with previous off-core approaches

Approaches	Target CPU	Multi-threaded support	Tagged memory	Kernel support	Tag bits	Tag scheme	Floating-point support
Kannan et al. [2]	Leon3 (softcore)	no	physical	partial	4	extended memory	no
Deng et al. [8], [9]	Leon3 (softcore)	no	physical	no	1-32	tag TLB	no
Heo et al. [3]	Leon3 (softcore)	no	physical	no	1	packed array (bitmap)	no
Lee et al. [7]	Leon3 (softcore)	no	physical	partial	1	packed array (bitmap)	no
Wahab et al. [5]	ARM Cortex-A9	no	N/A	no	1-32	address and tag	no
This work	ARM Cortex-A9	yes	virtual	yes	1-32	TMMU	yes

article, especially the communication between kernel and DIFT coprocessor.

Existing works target single-core CPUs. However, most of current CPUs are multi-core, even in embedded systems. As a consequence, the approach developed in this work is compatible with multi-threaded applications. To the best of our knowledge, no existing hardware approach implements DIFT for multi-threaded applications.

This paper is organized as follows. Section 2 provides insights on existing hardware DIFT solutions. Section 3 presents the proposed architecture and provides implementation details. Section 4 provides two case studies. Section 5 details implementation results and Section 6 gives some conclusions and future perspectives.

2. Related work and assumptions

Hardware-assisted DIFT has been a hot topic for the last decade. Important works have been done to implement efficient DIFT on FPGAs. There are four main types of hardware DIFT approaches proposed in the literature [5]:

- **Filtering hardware accelerators** [10], [11]. Instead of computing tags for each executed CPU instruction, this approach proposes to eliminate unmonitored events before computing and checking tags to lower DIFT time overhead.
- **In-core** approach [1], [4] modifies the architecture of the CPU to compute tags in parallel to regular operations.
- **Off-loading** approach [12] takes advantage of multi-core systems to offload tag computation on another general purpose core.
- **Off-core** approach [2], [5] consists of using a custom DIFT coprocessor to compute tags. The idea is similar to the offloading solution but instead of wasting a general purpose core, this approach uses a custom DIFT coprocessor.

Among the above solutions, only offloading and off-core DIFT approaches are portable to hardcores. The in-core solution modifies the hardware architecture and therefore is not feasible in practice. The offloading approach wastes a general purpose core for DIFT operations. Therefore, the off-core approach is the best candidate to implement DIFT for a hardcore.

Table 1 provides a comparison of this work with previous off-core approaches that use a dedicated co-processor to monitor the applications executed on the main CPU. Implementations for off-core approaches are generally done using softcores (e.g. Leon3 or SPARC V8 architecture) as the main CPU. In such solutions, the design is not easily portable on hardcores. Moreover, some designs do not target modern OS such as Linux. Therefore, it is necessary to overcome these limitations to implement DIFT.

This work is an improvement of [5] which proposes to use ARM CoreSight debug components, static analysis, and instrumentation to recover required information for DIFT on ARM SoCs. However, the solution proposed in [5] has several limitations. The coprocessor is implemented using a MicroBlaze softcore which limits its performance because the softcore needs to fetch instructions from memory, decode the instruction in software and then compute tags. Furthermore, it lacks kernel support, tagging of floating point and multi-threaded applications cannot be used.

The contributions of this paper are the following:

- Flexible security policies implementation in hardware: previous off-core solutions lack ways of specifying security policies (compile-time only or runtime only) and do not offer support for multiple security policies of different tag granularities (page, word, etc.).
- This work proposes to tag virtual addresses instead of physical addresses as done in related works. It offers the possibility to use the proposed approach with modern OS and hardcore CPUs with memory management units.
- This work is compatible with multi-threaded applications mainly thanks to the context ID information obtained from CoreSight PTM. It allows filtering trace using Context ID and to determine the exact order of operations on the CPU for each thread.
- This work explains how the Linux kernel communicates with the FPGA fabric available in a Zynq device (including a dual-core Cortex-A9 and reconfigurable logic): information missing in most, if not all, existing works. It shows how each information container is tagged and how the kernel synchronizes with the DIFT coprocessor.
- This work tracks all information flows unlike most existing works. It allows detecting a wider range of attacks, when compared to [5], including attacks targeting library code. Furthermore, the communication time overhead is improved by a factor 3.8 compared to existing work strategy, described in [3], with similar software and hardware constraints.

3. Proposed architecture

3.1. Overall architecture

Figure 1 shows the overall architecture of the proposed approach. The architecture is inspired from an existing off-core approach [5]. Instead of using a Microblaze as DIFT coprocessor as in [5], this work takes advantage of a custom DIFT coprocessor. Other main differences are flexible security

policies implementation, virtual memory tagging and support for multi-threaded software.

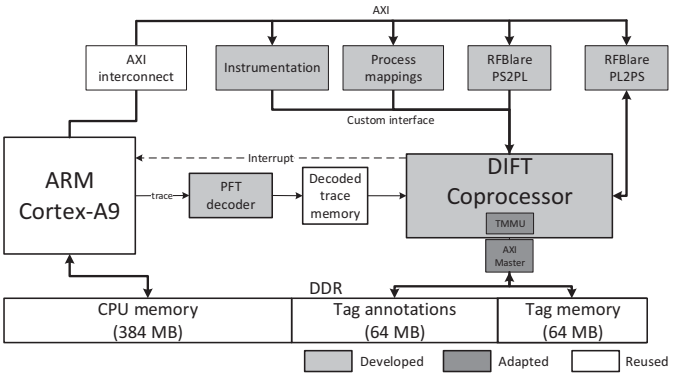


Figure 1: Overall system design with a DIFT coprocessor

This paragraph briefly explains how IFT operations are managed by the overall architecture. To begin with, the application is compiled using a custom embedded Linux distribution and SDK (*Software Development Kit*) generated using Yocto toolchain [13]. The modified compiler instruments the application binary to add `str` instructions as explained in Figure 4. During compilation, static analysis is done in order to generate annotations shown in Table 2. When the user launches the application, the Linux kernel loads application sections (e.g. `.text` section) in memory and annotations in tag annotations memory section. The kernel configures the CoreSight PTM to trace the entire `.text` section of the application. The trace comes out via the EMIO (*Extended Multiplexed Input/output*) interface in PFT (*Program Flow Trace*) protocol as described in [14]. The PFT decoder decodes the trace and stores it in the decoded trace memory. The decoded trace only contains information on starting address of basic blocks. What information flows happen inside a basic block needs to be determined. Static analysis allows to recover information flows for each basic block in the application and store them in tag annotations section of memory in a similar way as described in [5]. However, some information, such as `load`/`store` addresses, cannot be resolved statically. This missing information is obtained through instrumentation via Instrumentation IP (Figure 1). The DIFT coprocessor requires information from other modules such as process mappings IP which is explained in sections 3.4 and 3.7.3. RFBlare IPs, described in section 3.7.2, are used to communicate with the Linux kernel in order to get the tag of memory address or to set the tag of memory address. The DIFT coprocessor is responsible for managing these IPs, computing and storing tags in tag memory section.

3.2. Flexible security policies using DIFT coprocessor

The DIFT coprocessor has been designed in order to provide maximum flexibility in terms of security policy specification. The security policy specifies how to propagate and check tags in order to detect a particular kind of attack [15]. There are two missing aspects in existing works regarding security policies: flexibility to specify security policies and tag granularities.

Security policies can be specified at compile-time or at runtime. The compile-time solution [3] consists of hard-coding the propagation and checking operations using dedicated op-codes, during the compilation of the application. For instance, consider that the security policy states that all arithmetic and logic instructions on ARM core result in the logical `OR` operation on their corresponding tags. Then, during static analysis, for all arithmetic and logical instructions of the program code, an `OR` operation is hard-coded to compute the tags of their operands. The runtime solution [1] requires a special register called TPR (*Tag Propagation Register*) to specify the operation that has to be done on tag values. This time, the static analysis gives the operands for each instruction and the class of ARM instruction: Arithmetic/logical, Load/Store, Branch, Floating point Load/Store. Thanks to class information, the DIFT coprocessor can determine with the help of the TPR register, the operation that needs to be done to propagate the tags corresponding to the instruction executed on the main CPU. The main advantage is that the TPR value can be modified at runtime to modify the policy without recompiling the application.

No existing works provide the flexibility to specify security policies using both methods. This work proposes an architecture that can implement either one of these approaches providing developers more flexibility to implement security policies. The DIFT coprocessor ISA (*Instruction-Set Architecture*) has two different types of instructions: specific instructions for compile-time method and specific instructions for runtime method. The last ones are used in combination with TPR and TCR (*Tag Check Register*) [1] to implement a runtime security policy. Table 2 sums up different types of instructions (called annotations) supported by the custom DIFT coprocessor described in this work. There are four annotation types: Tag initialization, Tag ALU, Tag Load/Store and compound annotations. The same set of annotations is also included for the floating-point code. Tag initialization annotations can be used in both runtime and compile-time methods to initialize tags of registers or memory addresses. Tag ALU annotations propagate tags for registers. For instance, `TagRRR` annotation (on the fourth row of Table 2) shows a runtime annotation that contains `type` field and operands `T1, T2,` and `T3`. The operation `op` for this annotation will be determined, at runtime, by reading TPR register value for the corresponding `type`. If the `type` field is arithmetic and the TPR register states that an `AND` operation must be done on source operands to compute destination tag, then the operation done on TMC (*Tag Management Core*) unit of DIFT coprocessor is `T1 = T2 AND T3`.

Furthermore, existing off-core approaches do not provide different tag sizes and use a fixed tag size of one bit in most cases. Therefore, security policies that require multiple bits for a tag (such as heap overflow detection [15]) cannot be implemented. This work offers hardware support for a tag size up to 32 bits. In addition, it can support multiple security policies as discussed in Section 4.

3.3. Tag virtual memory

In related works, tags are associated with physical memory addresses. However, Linux applications are compiled to use virtual addresses and the MMU is responsible to translate them into physical addresses during execution. Existing solutions consider that it is possible to recover translation information from the MMU. This assumption is only realistic if the main CPU is a softcore tightly coupled with the DIFT co-processor. On a hardcore, a solution could be to modify the Linux kernel in order to send to the coprocessor information about PTEs (*Page Table Entries*), which are managed by the kernel. However, this can be costly because each time a translation is done, the virtual and physical page numbers need to be sent to the FPGA part as well. This information can be difficult to obtain from the kernel and needs lots of minor modifications to the kernel source code

TABLE 2: Overview of DIFT coprocessor (TMC) instructions (called annotations)

Instruction type	Opcode	Operation type	Example annotation	Action
Tag initialization	TagRImm	irrelevant	TagRImm T1,#1000	T1 = 1000
	TagRR	irrelevant	TagRR T2,T1	T2 = T1
	TagMR	irrelevant	TagMR R1,T1	Mem[R1] = T1
Tag ALU	TagRRR	runtime	TRR type T1,T2,T3	T1 = T2 op T3
	TagRRR2	compile-time	TagRRR2 AND T1,T2,T3	T1 = T2 AND T3
Tag Load/Store	TagMTR	runtime	TagMTR type R1,T1,#4	Mem [R1+4] = T1
	TagTRM	runtime	TagTRM type T1,R1,#4	T1 = Mem[R1+4]
	TagMTR2	compile-time	TagMTR2 R1,T1,#4	Mem[R1+4] = T1
	TagTRM2	compile-time	TagTRM2 T1,R1,#4	T1 = Mem[R1+4]
Compound	TagITR	runtime	TagITR T3,T1,#4	Mem[TMMU(Instrumentation)] = T3
	TagTRI	runtime	TagTRI T4,T2,#4	T4 = Mem[TMMU(Instrumentation)]
	TagITR2	compile-time	TagITR2 T12,#4	Mem[TMMU(Instrumentation + 4)] = T1
	TagTRI2	compile-time	TagTRI2 T2,#4	T2 = Mem[TMMU(Instrumentation + 4)]
	TagKTR	compile-time	TagKTR T1	Mem[RFblare_PL2PS] = T1
	TagTRK	compile-time	TagTRK T2	T2 = TMMU(Mem[RFblare_PS2PL])

which is constantly changing. In addition, the trace generated by the PTM contains virtual addresses. Therefore, to limit the kernel modifications and to avoid virtual to physical address translation overhead, a tag is associated to a virtual address rather than a physical address.

3.4. Tag memory management

We rely on a TMMU (*Tag Memory Management Unit*) to associate tags to a memory region used by the main CPU. The TMMU, see Figure 1) is functionally similar to an MMU used in CPUs. It translates each process virtual address to a physical tag address i.e. the physical address which contains the tag associated to the virtual address. It is implemented as an associative array of 64 entries with some additional logic. Each entry contains a virtual page number and the corresponding physical page number where the tag is located. In order to initialize the TMMU, the mappings of different segments of the process (e.g. code and data) are sent to the FPGA (Process mappings IP in Figure 1). This information is retrieved while the application is being loaded by the kernel and is sent to the FPGA part via process mappings IP (Figure 1). The process mappings IP contain 64 registers to store information for virtual address page numbers.

3.5. Coprocessor design

Figure 2 illustrates the high-level architecture of the DIFT coprocessor. The coprocessor has two main submodules: the dispatcher and the TMC unit.

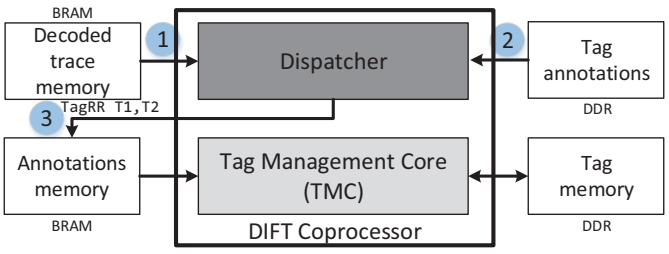


Figure 2: Coprocessor architecture

The dispatcher initializes and manages all IPs. Its main objective is to find annotations in program execution order (thanks to decoded trace) and to store them in local annotations memory for the TMC unit. The dispatcher is implemented as a classical five stages pipelined MIPS CPU (Figure 3) that allows executing general operations. It reads decoded trace (1), finds annotations corresponding to the decoded trace by reading tag annotations memory section (2) and stores them in local annotation memory (3).

The Figure 3 shows the internal architecture of the DIFT coprocessor. The TMC core is in charge of propagating and checking tags according to a security policy. The TMC core contains 3 register files: TRF, TRF_FP, and GRF. TRF contains tags corresponding to the main ARM core registers (`r0` to `r15`) while TRF_FP contains tags for the ARM floating point registers (`s0` to `s31`). GRF contains general purpose registers. If a compile-time security policy is being implemented, the tag check operation is performed by Tag ALU. Otherwise, at runtime, the tag check module is responsible for checking tags at the writeback stage. If a tag check fails, then an interrupt is sent to the ARM core to trigger a counter-measure (e.g. stopping the application).

The DIFT coprocessor managed two different types of data values: tag values, stored in TRF (*Tag Register File*) and general values, stored in GRF (*General Register File*). TRF is required in order to store tags of ARM CPU registers and GRF is required for general purpose computation. For instance, if a tag of memory address needs to be set to a value, then a general purpose register is needed to store the memory address.

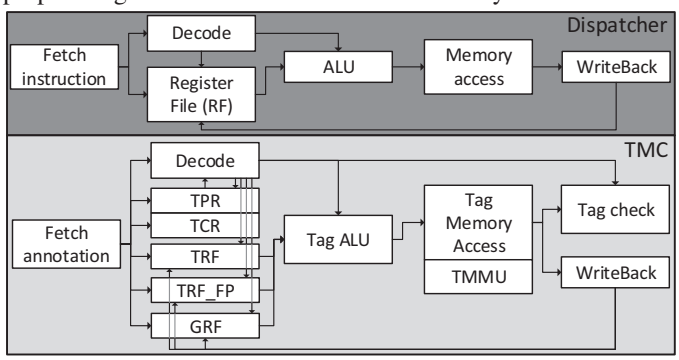


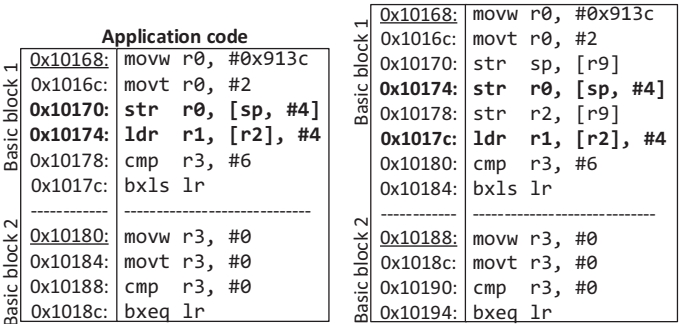
Figure 3: DIFT coprocessor microarchitecture

The TMC core is responsible for decoding annotations and computing specified tag operations either in the annotation itself or by using the security policy registers. It is a dedicated coprocessor for decoding annotations and is pipelined in five stages (Fetch, Decode, Execute, Memory Access, Write Back). It executes annotations that are detailed in Table 2. This core can be duplicated in order to deal with multiple security policies or in order to propagate tags for multiple threads or processes.

3.6. Multi-thread support

To the best of our knowledge, no existing hardcore solutions support multi-threaded applications. The main issue in existing

architectures is how the FPGA can determine which thread is running on the main CPU. This work proposes to take advantage of the trace generated by the PTM in order to determine the exact order of the thread execution. The ARM CoreSight PTM can be configured, providing OS support, to recover the TID (*Thread ID*) of threads on each context switch. By filtering the trace with TID, the execution order of threads can be determined. As the trace itself respects the execution order of each thread [5], the CFG (*Control Flow Graph*) of each thread can be determined. In other words, the DIFT coprocessor can determine, thanks to the context ID feature, the thread executed on the ARM CPU and can propagate tags for the corresponding information flows. As each thread requires a dedicated TMC unit, the number of threads supported by the proposed architecture is limited by the number of TMC units that can be implemented in FPGA.



(a) Original application (b) Instrumented application

Figure 4: Binary instrumentation using modified LLVM

3.7. Software requirements

All existing off-core approaches require software modifications. However, existing works do not provide enough information to easily reproduce their work. Therefore, in this subsection, all proposed software modifications are explained in order to make this work reproducible.

3.7.1. Instrumentation. The DIFT coprocessor can reconstruct the program execution path by creating a CFG using the decoded trace. However, it has no information on what happens inside each CFG basic block. The output of the static analysis, as done in [5], states how to propagate tags for each CPU instruction. However, for memory instructions, the static analysis cannot compute the memory addresses that are only known at runtime.

This situation is illustrated in Figure 4. Figure 4a shows the original application code. The starting address of basic blocks 1 and 2 (underlined addresses) are recovered from the decoded trace. For memory instructions (in bold in Figure 4a) such as `ldr` and `str` at addresses `0x10170` and `0x10174` of the original application, the value of registers (`sp` and `r2`) cannot be computed from static analysis. This information is needed by the DIFT coprocessor to propagate tags from a register to a memory address (in case of a store operation) or from a memory address to a register (in case of a load operation).

In this work, this information is obtained by instrumenting the original application binary. The instrumented application is shown in Figure 4b. Before each memory instruction, another instruction is added (store instructions at addresses `0x10170` and `0x10178` of Figure 4b) that sends the missing register value to the memory address contained by `r9`. The `r9` register has been reserved and is not used by the application. It contains the virtual address associated with the physical address of instrumentation IP (shown in Figure 1) such that any store to `r9` results in a write to the instrumentation IP.

3.7.2. Kernel support. Existing hardware DIFT approaches only handle tags associated to RAM and registers. However, they do not take into account information stored on mass storage, i.e. tags associated to files. This feature is important to handle inter-process communications and data persistence (after a reboot). Moreover, users are more inclined to specify a policy at the file level rather than at the memory address or register level. For example, a user can easily identify the files that are supposed to contain confidential information, such as passwords.

Handling file implies some kernel support. However, the major kernel modifications are limited to file Input/Output interface. Only a limited number of system calls such as `read` or `write` have to be modified. To handle tags associated to files, we rely on RFBlare [16]. RFBlare is a modified Linux kernel that implements an OS-level DIFT monitor. This monitor saves tags as file meta-data using file system extended attributes. RFBlare original behavior consists in propagating tags from files to the process memory whenever a file is read and from memory to files whenever a file is written. This OS-level approach is coarse-grained since only one tag is used to abstract the whole memory of a process.

We modify RFBlare to disable the OS-level tag propagation and we only used the feature that allows associating tags to files. We add some code to enforce a communication between RFBlare and the DIFT co-processor for each file I/O. For instance, when a `read` system call occurs on the ARM core, RFBlare allows retrieving the tag of the file being read, the address of the buffer where the read data is stored and the number of bytes read. These three values are sent by the kernel to the FPGA part using RFBlare_PS2PL FIFO IP (shown in Figure 1). Similarly, if a program writes to a file (e.g. `write` system call), the kernel sends the memory address of the buffer being written and the size of the buffer. Then, the DIFT coprocessor fetches the corresponding tag and sends it back to the kernel. RFBlare uses this tag to set the new tag of the file. All the communication for the `write` system call use RFBlare_PL2PS FIFO IP (shown in Figure 1). These FIFOs have an AXI-Lite interface to communicate with the ARM core and a custom FIFO interface to communicate with the TMC.

The main CPU and the DIFT coprocessor has to be synchronized since the main core runs faster than the DIFT coprocessor on FPGA. In this work, this synchronization is done thanks to the RFBlare_PL2PS and RFBlare_PS2PL FIFO mechanisms. As the ARM core and the DIFT coprocessor runs on different frequencies, the attack might be detected after execution of the malicious code. However, it will not compromise the system because an attack needs a system call (for example `write`) to damage the system. However, each system call waits for the DIFT coprocessor to finish tag computation before carrying on execution. Therefore, the synchronization mechanism makes sure that the software attack does not affect the system.

3.7.3. Process mappings. The DIFT coprocessor needs to know memory mappings of the program in order to properly initialize tag MMU. We modified the ELF binary loader (`binfmt_elf.c`) to send memory mapping of the process. This way, the DIFT coprocessor starts by initializing TMMU before any other operation is done.

4. Case studies

This section shows two use cases where the proposed architecture can outperform existing works: handling multiple security policies and multi-threaded applications.

This work takes advantage of the internal architecture of the DIFT coprocessor to reuse developed modules. Figure 5 shows the proposed architecture for analyzing two threads. The hardware consists of a single dispatcher and two TMC units. This design can also be used to configure multiple security policies. This work proposes to duplicate some modules of the DIFT coprocessor to track multiple threads. The other possible solution is to use context switch i.e. each time the ARM processor switches to another thread, the context of DIFT coprocessor is saved and the new context is restored. However, this solution will add an important storage overhead and time overhead because it would require storing the entire content of three register files and other important CPU registers.

4.1. Multiple security policies

In order to implement multiple security policies of different tag sizes, Dhawan et al. proposed to modify the internal architecture of the CPU to handle data and tag [4]. Their approach relies on a 128-bit CPU with 64-bit of data and 64-bit of tag. However, their architecture is not modular and flexible. For instance, the area and power overhead remain important even if we consider only a single security policy.

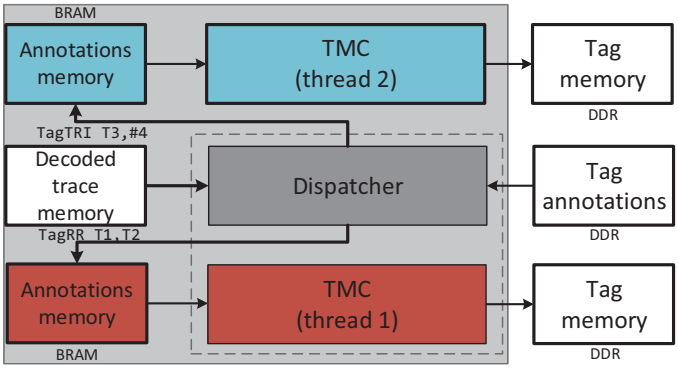


Figure 5: Coprocessor architecture for two threads

The architecture illustrated by Figure 5 can be used for implementing multiple security policies. In this case, the thread 1 core executes one security policy and the thread 2 core executes another security policy. The dispatcher reconstructs the execution path of the application and stores annotations in both annotations memories. Then, each TMC unit propagates tags according to its own security policy. If a violation occurs in one of the TMC units, an interrupt is sent to the ARM core to stop execution. This approach is modular since we can adapt the number of TMC units to the number of policies to verify in parallel.

4.2. Multiple threads

Real-world applications use multiple threads in order to speed up execution. However, existing DIFT mechanisms are not able to track multiple threads because the interface between the main processor and the DIFT coprocessor does not export information that allows determining the thread being executed on the CPU. This approach exploits CoreSight components to extract the context ID, which comprises the TID (*Thread ID*) and the ASID (*Application-Specific ID*).

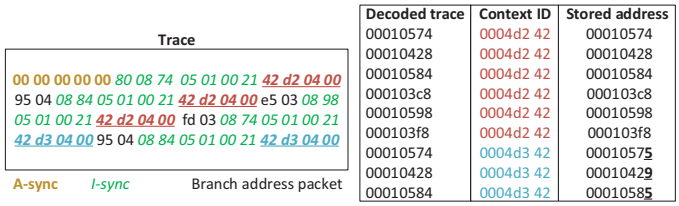


Figure 6: Decoded trace for multiple threads

Single core. If two threads are considered and both run on the same CPU core, the context ID field, retrieved by decoding the PTM trace, allows to determine which thread is currently being executed. Figure 6 shows the format of decoded traces in memory. The trace contains I-sync packet (in green) which contains 4 bytes of context-ID (underlined). It can be noticed that the trace contains the same ASID value (`42`) and two different TID values (`4d2` and `4d3`) corresponding to each thread.

The PFT decoder computes the branch address where the ARM core has jumped. These addresses are 4-byte aligned (in ARM state which is the only state allowed to be used during compilation) which means that two bits (0 and 1) are always equal to 0. The PFT decoder also recovers the context ID field and uses these unused two bits to specify whether the decoded trace is generated by the first thread or the second thread. Figure 6 shows that for the second thread, the stored addresses in decoded trace memory are not 4-bit aligned due to the storage of context ID in last two bits. The value of the context ID still needs to be stored in separate registers so that if an attack is detected, the interrupt routine gets the TID of the program in order to kill the process in charge of generating unauthorized behavior.

Multi-core system. If two threads are launched on two different CPUs, the same architecture can be used to propagate tags. However, the trace configuration and the PFT decoder needs to be adapted. In this experiment, we only monitor a single ARM Cortex-A9 core. If the second core is to be considered, the second PTM needs to be configured as well to trace the program. In terms of configuration, when both PTMs are enabled, each PTM needs to insert a trace ID packet so that the funnel [17] can merge traces onto a single bus. Furthermore, trace sinks (TPIU or ETB) must enable formatting to differentiate trace from different sources. Therefore, the PFT decoder should be adapted in order to consider formatted data instead of raw data considered in this work. The rest of the architecture remains the same as it is similar to multiple threads on a single core case. For multi-core case, a tracing overhead will appear due to trace formatting that adds low overhead of 6% [18] and an overhead of one byte every time the trace bus switches between trace sources.

5. Implementation Results

Xilinx tools 2017.1 are used on a Xilinx Zedboard with a Z-7020 SoC (dual-core Cortex-A9 running at 667 MHz and an Artix-7 FPGA) to implement the architecture shown in Figure 1. The Clang compiler has been used with customized LLVM pass to get the binary and annotations (instructions for TMC). The evaluation described in this section has the following goals:

- Evaluate feasibility of proposed architecture.
- Evaluate the cost of software modifications (execution time overhead and memory footprint).
- Compare efficiency with related works.

5.1. Area results

Table 3 shows the area overhead of this work. Most of the FPGA area is filled by AXI interconnect (5.14%), dispatcher (4.18 %) and TMC (3.45%).

TABLE 3: Post-synthesis area results on Xilinx Zynq Z-7020

IP Name	Slice LUTs (in %)	Slice Registers (in %)	BRAM Tile
Dispatcher	2223 (4.18%)	1867 (1.75%)	3
TMC	1837 (3.45%)	2581 (2.43%)	6
PFT Decoder	121 (0.23%)	231 (0.22%)	0
Instrumentation	676 (1.27%)	2108 (1.98%)	0
Blare PS2PL	662 (1.24%)	2106 (1.98%)	0
Blare PL2PS	62 (0.12%)	56 (0.05 %)	0
Decoded trace memory	0	0	2
AXI Master	858 (1.61%)	2223 (2.09 %)	0
TMMU	295 (0.55%)	112(0.10 %)	3
AXI Interconnect	2733 (5.14%)	2495 (2.34 %)	0
Miscellaneous	1381 (2.6%)	2160 (2.03%)	0
Total Design	**10848 (20.39%)**	**15939 (14.98%)**	**14 (10%)**
Total Available	**53200**	**106400**	**140**

The overall design takes 20.4% of the FPGA area. If two security policies are required at the same time, the design would be modified as shown in Figure 5. The overall design, in case of two security policies, would take additional 4095 slice LUTs, 9074 slice registers (i.e. 8% additional FPGA logic) and 6 BRAM tiles. In other words, the proposed design can run more than 8 security policies or protect more than 8 processes at the same time.

5.2. Time overhead analysis

CoreSight components do not add any execution time overhead [5]. Even though this work uses a different configuration of CoreSight components since the context ID tracing is enabled, CoreSight components still do not add any noticeable execution time overhead. The static analysis does not add time overhead because it is performed offline, during compilation. Therefore, the timing overhead of the proposed approach is only due to instrumentation.

The overhead of instrumentation is analyzed by measuring instrumented application execution time normalized to the original application (non-instrumented) running time. Execution times are measured using a set of custom applications, such as FFT (Fast Fourier Transform) and CRC (Cyclic Redundancy Check) computations, on the Linux kernel 4.9, patched with CoreSight TPIU driver, using the Linux `perf` command.

Figure 7 shows the average normalized execution time overhead for instrumented application binaries. The related work implementation strategy is detailed in [3]. Two instrumentation strategies (strategy 1 and 2) are adapted from [5]. However, the difference between their proposed strategies and this work resides in static analysis, instrumentation and hardware modules. All information flows are considered in this work unlike [5]. Library code is instrumented unlike [5] and different configuration of Coresight components is used which changes the design of hardware modules such as PFT decoder.

Figure 7 shows that if related work strategy as in [3] is used, the instrumentation overhead, on average, is 12.79 times higher than the original execution time. The instrumentation strategy 1, in which all memory instructions are instrumented, adds significant execution time overhead (on average 10.43 times higher than the original execution time). The instrumentation overhead is high for applications that require more memory operations (such as `lu` and `matrix`). However, if only register-relative (other than `PC, SP,` and `FP`) memory instructions are instrumented, as in strategy 2, the average communication time overhead is reduced by a factor 3.8 to achieve 3.35 times higher execution time on average. The three main reasons why this overhead remains high are: this work targets hardcore CPU which is not the case in most existing works such as in [1], [3], [4], [8], the static analysis considers all information flows rather than function level information flows as in [3], [5] and it also instruments library code used by the applications unlike [5].

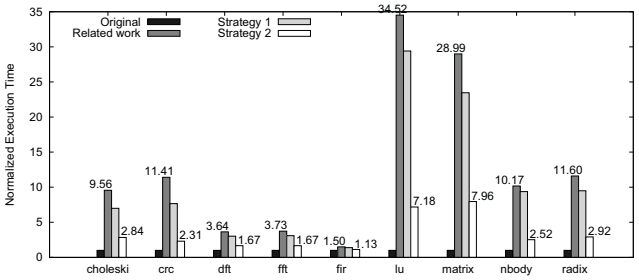


Figure 7: Average execution time overhead for custom benchmark

5.3. Memory Footprint

Figure 8 shows the memory space overhead of custom benchmark applications using both strategies. Binaries are statically compiled i.e. all the code executed by the application is inside the code section of the binary. Therefore, results obtained here take into account all modifications of user code and the library code. The instrumentation strategy 1 adds in average 10% of memory space overhead and instrumentation strategy 2 adds in average only 3% of memory space overhead.

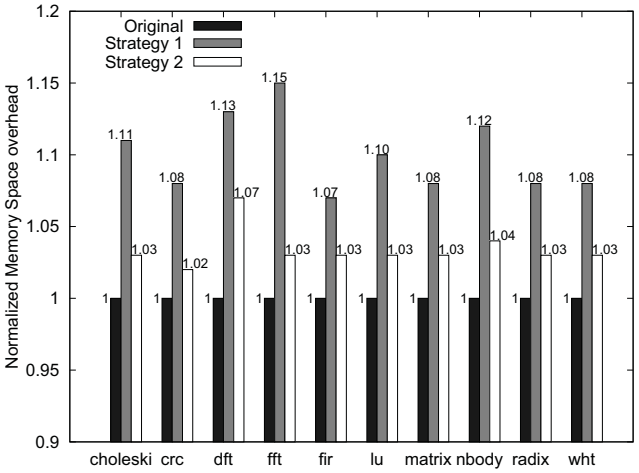


Figure 8: Memory footprint of custom benchmark

5.4. Comparison with related works

Table 4 shows the comparison of an off-core approach like this work with previous off-core approaches. Comparison of this work with in-core solutions is not done because they are not hardcore portable and are very invasive contrary to the off-core solution used in this work.

Compared to [5], this work has higher area overhead because it provides support for FP and multi-threaded software that requires additional modules in the FPGA area. Furthermore, the power overhead of this work is similar to [5] because this work does not use any DSPs unlike the coprocessor used in [5] which

TABLE 4: Performance comparison with previous off-core approaches

Approaches	Kannan [2]	Deng [8]	Heo [3]	Heo [3] adapted	Wahab [5]	This work
Area overhead	6.4%	14.8%	14.47%	N/A	**0.47%**	0.95 %
Power overhead	N/A	**6.3%**	24%	N/A	16%	16.2%
Max frequency	N/A	**256 MHz**	N/A	N/A	250 MHz	250 MHz
Communication time overhead	N/A	N/A	60%	1280%	**5.4%**	335%
Hardcore portability	No	No	**Yes**	**Yes**	**Yes**	**Yes**
Main CPU	Softcore	Softcore	Softcore	**Hardcore**	**Hardcore**	**Hardcore**
Library instrumentation	N/A	N/A	partial	**Yes**	No	**Yes**
All information flows	No	No	No	**Yes**	No	**Yes**
FP support	No	No	No	No	No	**Yes**
Multi-threaded support	No	No	No	No	No	**Yes**

is a MicroBlaze with DSP units that are more power consuming than normal FPGA logic. The maximum frequency achievable is comparable to existing works.

The most important difference resides in communication time overhead. It may appear that this overhead is higher than values reported in related works. The fifth column, Heo [3] adapted, shows that if the proposed instrumentation strategy in related work [3] is adapted with the same static analysis and information flows constraints as in this work, then the communication time overhead obtained on Zynq architecture reaches 1280%. However, the proposed solution in this work can reduce this overhead to 335% by a factor $\frac{1280}{335} = 3.8$. The best communication time overhead reported in existing works is 5.4 % ([5]) but their proposed solution lacks support for all information flows and library instrumentation. Furthermore, the reported value is an estimation based on the number of instrumented instructions whereas the value reported in this work is obtained on the Zedboard platform and the time is measured using `perf` tool on Linux kernel v4.9. It is very important to consider all information flows and instrument libraries because it allows detecting an important range of attacks unlike most existing works. For instance, a simple attack on existing works could be to add a wrapper around library function and use Linux kernel dynamic `LD_PRELOAD` feature to avoid detection of any malicious library code. However, this work is able to detect the execution of malicious library code as its tracking is not ignored as in most existing works. This work provides support for floating point (through additional instructions and additional register file) and multi-threaded software thanks to context ID feature of the CoreSight PTM as described in section 4: support for these features is missing in related works.

6. Conclusion

This work is the first work providing support and flexibility to implement multiple security policies which can either be specified at run-time or compile-time. It provides features missing in related works: protection of floating point code and multi-threaded software. Both features are essential in protecting real-world applications. This work takes advantage of CoreSight components along with LLVM modifications to implement information flow security policies. The communication time overhead is reduced to more than 380% if compared to existing work strategy [3]. Area results show interesting perspectives in terms of implementing multiple security policies and protecting multiple processes. For instance, protecting two threads requires another TMC unit which adds an additional area overhead of 8%. As this work deals with multi-threaded applications that do not share memory, the next step will be to include support for threads that share memory. Another improvement would be adding support for more than 8 threads by making context switches on each TMC unit.

Acknowledgments

This work has received a French government support granted to the COMIN Labs excellence laboratory and managed by the National Research Agency in the “Investing for the Future” program under reference ANR-10-LABX-07-01.

References

[1] M. Dalton, H. Kannan, and C. Kozyrakis, “Raksha: A flexible information flow architecture for software security,” vol. 35, no. 2. New York, NY, USA: ACM, Jun. 2007, pp. 482–493.

[2] H. Kannan, M. Dalton, and C. Kozyrakis, “Decoupling dynamic information flow tracking with a dedicated coprocessor,” in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, Jun. 2009, pp. 105–114.

[3] I. Heo, M. Kim, Y. Lee, C. Choi, J. Lee, B. B. Kang, and Y. Paek, “Implementing an application-specific instruction-set processor for system-level dynamic program analysis engines,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 4, pp. 53:1–53:32, Sep. 2015.

[4] U. Dhawan, C. Hritcu, R. Rubin, N. Vasilakis, S. Chiricescu, J. M. Smith, T. F. Knight, Jr., B. C. Pierce, and A. DeHon, “Architectural support for software-defined metadata processing,” *SIGARCH Comput. Archit. News*, vol. 43, no. 1, pp. 487–502, Mar. 2015.

[5] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapôtre, and G. Gogniat, “Armhex: A hardware extension for dift on arm-based socs,” in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–7.

[6] ARM, *Embedded Trace Macrocell Architecture Specification*.

[7] J. Lee, I. Heo, Y. Lee, and Y. Paek, “Efficient security monitoring with the core debug interface in an embedded processor,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 8:1–8:29, May 2016.

[8] D. Y. Deng, D. Lo, G. Malysa, S. Schneider, and G. E. Suh, “Flexible and efficient instruction-grained run-time monitoring using on-chip reconfigurable fabric,” in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 137–148.

[9] D. Y. Deng and G. E. Suh, “High-performance parallel accelerator for flexible and efficient run-time monitoring,” in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, Jun. 2012, pp. 1–12.

[10] S. Fytraki, E. Vlachos, O. Kocberber, B. Falsafi, and B. Grot, “Fade: A programmable filtering accelerator for instruction-grain monitoring,” in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 108–119.

[11] S. Chen, M. Kozuch, T. Strigkos, B. Falsafi, P. B. Gibbons, T. C. Mowry, V. Ramachandran, O. Ruwase, M. Ryan, and E. Vlachos, “Flexible hardware acceleration for instruction-grain program monitoring,” in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 377–388.

[12] V. Nagarajan, H.-S. Kim, Y. Wu, and R. Gupta, “Dynamic information flow tracking on multicores,” *Workshop on Interaction between Compilers and Computer Architectures, (colocated with HPCA)*, Feb. 2008.

[13] Yocto project, *Yocto Project Reference Manual*.

[14] ARM, *CoreSight Program Flow Trace Architecture Specification*.

[15] A. A. d. Amorim, M. Dénès, N. Giannarakis, C. Hritcu, B. C. Pierce, A. Spector-Zabusky, and A. Tolmach, “Micro-policies: Formally verified, tag-based security monitors,” in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 813–830.

[16] L. Georget, M. Jaume, G. Piolle, F. Tronel, and V. V. T. Tong, “Information flow tracking for linux handling concurrent system calls and shared memory,” in *Software Engineering and Formal Methods*, A. Cimatti and M. Sirjani, Eds. Cham: Springer International Publishing, 2017, pp. 1–16.

[17] ARM, *CoreSight Components Technincal Reference Manual*.

[18] ARM, *CoreSight Technology System Design Guide*.

14 Rejeu matériel d'instructions face aux attaques par injection de fautes

Cette section présente les travaux de recherche réalisés dans le cadre de la thèse de Mme Noura Ait Manssour débutée en 2019. Ces travaux s'inscrivent dans le cadre de la protection des architectures de processeurs embarqués face aux attaques physiques. Plus particulièrement, nous nous sommes concentrés sur les attaques par injection de fautes ciblant le flot de données.

14.1 Les attaques physiques par injection de fautes

Ce type d'attaques physiques consiste à perturber le fonctionnement d'un circuit dans le but d'engendrer des erreurs d'exécution pouvant mener à un état exploitable par l'attaquant. En 1997, l'article [Boneh 1997] a introduit théoriquement l'attaque par injection de fautes matérielles comme un nouveau moyen permettant d'extraire les secrets manipulés dans des cryptosystèmes asymétriques tels que RSA en exploitant des calculs erronés ou des valeurs de registre corrompues. Cette attaque repose sur le fait que le matériel n'a aucun indicateur de fautes injectées par l'attaquant. Depuis, de nombreuses attaques par injection de fautes ont été développées dans le but d'extraire des données sensibles [Boneh 2001][Dehbaoui 2013], de contourner des procédures d'authentification [Dureuil 2016] ou d'obtenir plus de privilèges [Timmers 2016].

Pour réaliser des injections de fautes au sein de circuits, différentes techniques ont été proposées [Bar-El 2006]. Les plus courantes sont la perturbation d'horloge, la perturbation de la tension d'alimentation, le rayonnement lumineux par laser et l'injection d'impulsions électromagnétiques. Il est important de noter que la précision en terme de localisation spatiale et temporelle varie pour chacune de ces techniques d'injection. De plus, chacune requiert une instrumentation matérielle particulière et une configuration dont le coût dépend du type de technique. Les perturbations d'horloge sont des variations locales de la fréquence d'horloge menant à la violation temporaire de la fréquence maximale supportée par le circuit. Cela peut par exemple, mener à des erreurs lors de transferts mémoires. Pour mettre en oeuvre ce type d'injection, l'accès au signal d'horloge est nécessaire. Une illustration de cette méthode ciblant l'algorithme AES implémenté sur FPGA est proposée dans [Agoyan 2010]. Les perturbations de la tension d'alimentation consistent à faire varier la tension d'alimentation d'un circuit cible par rapport à sa tension nominale. En effet, une sous- ou sur-alimentation du circuit a un impact direct sur les temps de propagation. La manipulation temporaire de l'alimentation par l'attaquant permet alors de violer certaines contraintes temporelles du circuit menant à des traitements erronés. L'article [Bozzato 2019] illustre l'utilisation de ce type d'injection pour extraire les *firmwares* de six microcontrôleurs. Les injections d'impulsions électromagnétiques utilisent un champ électromagnétique à la surface du circuit cible afin de générer des courants sur un ensemble de pistes et ainsi perturber le fonctionnement du dispositif. En 2002, l'article [Quisquater 2002] a introduit ce moyen d'attaque en décrivant l'usage d'une sonde induisant un fort champ magnétique transitoire sur un microprocesseur. Cette technique d'injection est particulièrement populaire car elle ne nécessite pas d'accès au signal d'horloge ou à l'alimentation du circuit et peut être réalisée sans décapsulation du circuit. Les articles [Dehbaoui 2012] [Dureuil 2015] [Trouchkine 2021] illustrent l'utilisation de cette technique pour mener des attaques ciblant des algorithmes cryptographiques dans divers contextes. Les injections de lumière focalisée, consistent généralement à injecter une faute avec un laser dans un circuit. En effet, le rayonnement lumineux peut provoquer la conductivité de transistors cibles et ainsi perturber le fonctionnement du circuit. Cette méthode a été adoptée pour la première fois dans [Skorobogatov 2003]. Cette technique est plus précise et efficace que les injections électromagnétiques mais nécessite une décapsulation du circuit. L'article [Vasselle 2017] illustre cette approche pour mettre à mal le démarrage sécurisé d'un smartphone. Il est intéressant de noter que des travaux récents ont démontré la faisabilité d'utiliser jusqu'à quatre faisceaux laser simultanément permettant la réalisation de scénarios d'injections complexes [Colombier 2021].

14.2 Le rejeu matériel d'instructions

Parmi les contre-mesures existantes pour lutter contre les attaques par injection de fautes [Karaklajić 2013][Bar-El 2006], la redondance temporelle est classiquement mis en oeuvre pour protéger des applications sensibles s'exécutant sur un processeur. Elle permet de protéger le flux de données à travers des schémas de tolérance aux fautes [Moro 2014] ou de détection de fautes [Barenghi 2010]. La redondance temporelle logicielle peut s'appliquer à différents niveaux : au niveau du programme ; au niveau de l'algorithme ; au niveau des instructions. Les travaux présentés dans cette section se sont focalisés sur la redondance au niveau instruction. Les articles [Barenghi 2010] et [Thati 2019] présentent deux catégories de cette forme de protection en logiciel à savoir la duplication d'instructions avec comparaison et la triplication d'instructions avec vote majoritaire. Cependant, ces solutions purement logicielles engendrent des coût en terme de temps d'exécution ($\times 2$ à $\times 4$) et de taille de code ($\times 2$ à $\times 14$ par instruction protégée) pouvant être prohibitif [Barenghi 2010].

Pour palier à ces limitations, nous avons développé et évalué des protections s'appuyant sur le rejeu matériel d'instructions. Deux stratégies ont été proposées : le rejeu avec comparaison et la triplication avec vote majoritaire. Pour la première stratégie, différentes versions sont proposées. Tout d'abord, nous avons proposé une version d'un processeur qui ne gère qu'un nombre de rejeux fixé dans le code assembleur. La figure 27 illustre cette stratégie (côté droit de la figure) et la compare à une approche purement logicielle (côté gauche de la figure). Dans le cas de la protection matérielle proposée, le jeu d'instructions du processeur a été étendu pour intégrer l'instruction ***rpl*** n w où n représente le nombre de fois qu'une instruction doit être rejouée et w la taille de la fenêtre contenant les instructions à protéger suivant l'instruction *rpl*. Dans le cas de la figure 27, les trois instructions I1, I2 et I3 seront ré-exécutées une fois. En comparant la solution avec une duplication logicielle, on remarque que le code assembleur résultant ne nécessite pas l'intégration de branchements supplémentaires réalisant la comparaison des résultats. En effet, dans l'approche proposée, un élément matériel dédié intégré au processeur réalise cette opération.

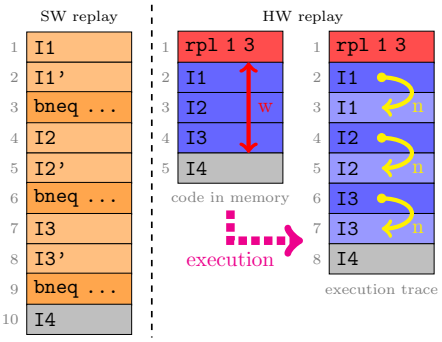


FIGURE 27 – Illustration de la stratégie de rejeu d'instructions matériel avec n fixé et comparaison avec la duplication logicielle

Dans un deuxième temps, nous avons proposé une version permettant un nombre aléatoire de rejeux pour réduire les chances pour un attaquant de maîtriser précisément l'instant d'injection. Ce rejeu aléatoire possède deux modes de fonctionnement : Le nombre de rejeux n est aléatoire et uniforme pour toutes les instructions de la fenêtre d'instructions protégées (RRCU) ; le nombre de rejeux n est aléatoire et variable pour chaque instruction de la fenêtre d'instructions protégées (RRCV). La figure 28 illustre ces deux modes en considérant une fenêtre w de taille 3. L'instruction *rpl* a été modifiée afin de définir le mode de fonctionnement. De plus, pour mettre en oeuvre ces modes de fonctionnement un PRNG (TRIVIUM), a été couplé au processeur cible.

Pour la deuxième stratégie, nous avons proposé des versions de processeur en ajoutant la triplication avec vote majoritaire (TV) afin d'obtenir un gain significatif en termes de temps d'exécution et de taille de code par rapport à la triplication logicielle. La figure 29 illustre cette

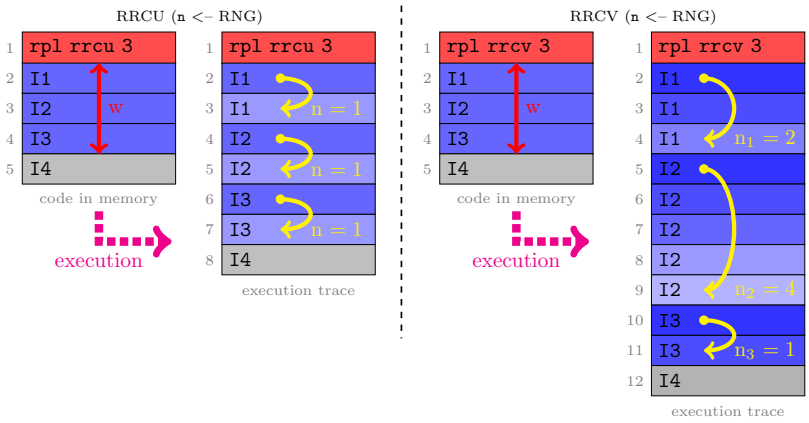


FIGURE 28 – Illustration des modes de fonctionnement RRCU et RRCV

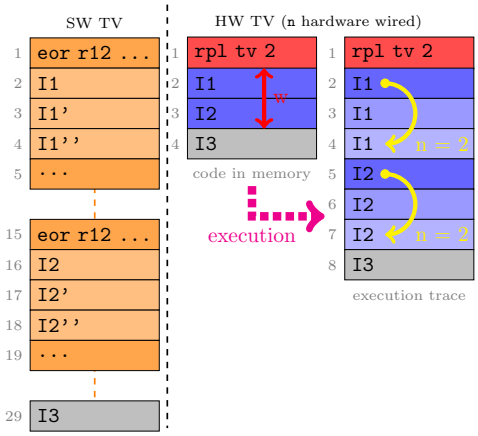


FIGURE 29 – Illustration de la stratégie de triplication avec vote majoritaire et comparaison avec l'approche logicielle équivalente

stratégie en considérant une fenêtre w de taille 2. L'instruction *rpl* a été modifiée afin de définir la stratégie utilisée. En comparant la solution à une approche équivalente logicielle nécessitant le replacement de chaque instruction protégée par environ 14 instructions [Barenghi 2010], on observe une réduction significative de la taille du code assembleur résultant et du nombre d'instructions à exécuter puisque seule l'instruction *rpl* est ajoutée au code original.

14.3 Conception d'un processeur intégrant un mécanisme de rejeu matériel d'instructions

Les solutions présentées précédemment ont été implémentées dans un processeur RISC 32 bits, développé pour ces travaux en System Verilog. Ce processeur est proche d'un coeur RISC-V RV32IM simplifié et possède deux étages de pipeline (fetch | decode + execute). Le banc de registre possède 32 registres de 32 bits et possède deux ports de lecture et 1 port d'écriture. Le registre R0 est câblé à la valeur zéro. Le processeur contient deux unités de calcul : une unité arithmétique et logique (ALU) et un multiplieur. L'unité LOAD/STORE (LSU) possède un accès direct à la mémoire de données (pas de hiérarchie de cache). Le processeur inclut également un ensemble de compteurs de performance pour le nombre de cycles, d'instructions, d'accès mémoires, de branchements, et pour la surveillance des protections.

Le jeu d'instructions de ce processeur a été étendu pour intégrer l'instruction *rpl* décrite dans la section 14.2. L'architecture de processeur proposée permet un nombre de rejeux n maximal de 4 (codé sur 2 bits) et une taille de fenêtre contenant les instructions à protéger w maximale

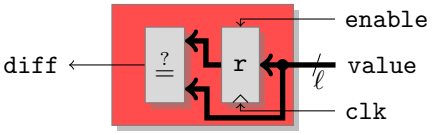


FIGURE 30 – Module matériel de détection de faute pour la stratégie de rejeu avec comparaison

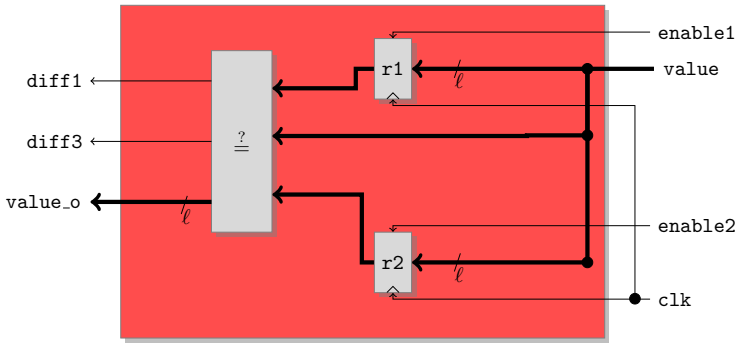


FIGURE 31 – Module matériel de vote pour la stratégie de triplication avec vote

de 16 (codée sur 4 bits). Durant l'exécution d'instructions protégées, des modules de protection matériels dédiés sont utilisés pour comparer les résultats des traitements réalisés. Trois modules ont été développés durant les ces travaux. Le premier, présenté en figure 30, compare pour chaque rejeu la valeur courante avec la valeur stockée lors de la première exécution de l'instruction rejouée. Un signal *diff* est levé en cas d'inégalité lors de la comparaison, signalant une faute détectée. Ce module est utilisé pour mettre en oeuvre la stratégie de rejeu avec comparaison.

Le second module, présenté en figure 31, met en oeuvre un vote majoritaire basé sur trois rejeu d'une même instruction. Deux registres sont utilisés pour stocker les valeurs issues des deux premières exécutions de l'instruction. À l'issue de la troisième exécution, un vote majoritaire est réalisé. Le signal *diff1* indique au contrôleur du processeur qu'une valeur parmi les 3 est différente et que la faute a été "absorbée" par la protection. Le signal *diff3* indique que les trois valeurs sont différentes et que les traitements doivent être interrompus.

Le troisième module, présenté en figure 32, rassemble les deux modules précédents. Une entrée dédiée de ce module nommée *mode* permet de sélectionner le mode de fonctionnement. Intégrer ce module au sein d'un processeur permet d'adapter la protection, à l'exécution, en accord avec la politique de sécurité.

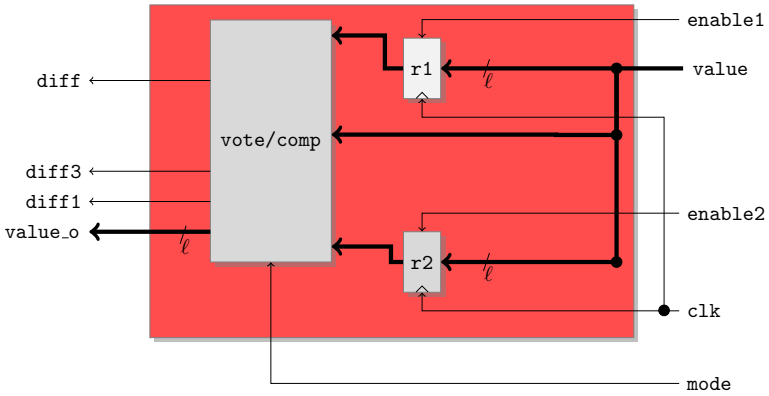


FIGURE 32 – Module matériel de détection et de vote pour le support des stratégies de rejeu avec comparaison et de triplication avec vote

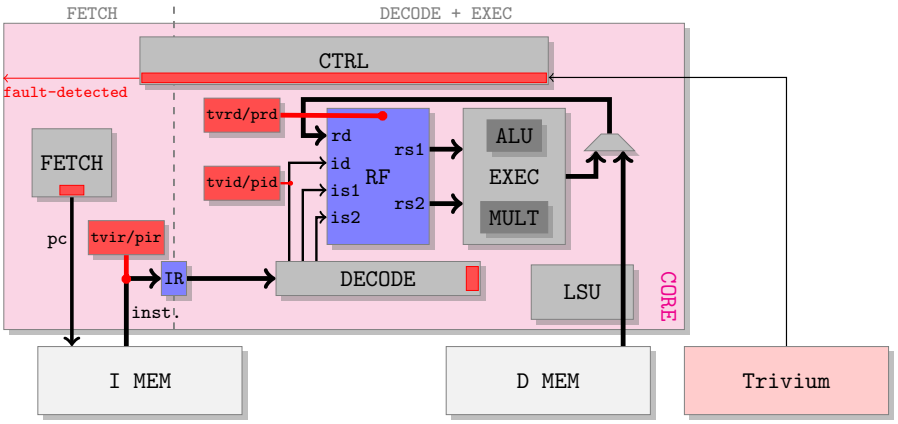


FIGURE 33 – Mise en oeuvre des modules de protection matériel au sein du processeur cible

Dans le but de protéger le flot de donnée au sein du processeur cible, ces différents modules peuvent être placés à de multiples emplacements. La figure 33 illustre un exemple d'intégration de trois modules de protection au sein du processeur cible. Dans cet exemple, un premier module (*tvid/pid*), est placé en amont du banc de registres *RF* pour protéger l'index du registre destination *id* pendant la phase de rejeu d'une instruction. Un second module (*tvrd/prd*) permet de protéger le résultat des calculs ou des lectures en mémoire réalisées dans l'étage d'exécution. Un troisième module (*tvir/pir*) est placé en amont du registre d'instruction *IR*. Via la protection du registre d'instruction, ce module protège tous les champs d'une instruction, y compris les index sources et destination du banc de registres. Le module (*tvid/pid*) est alors redondant.

La figure 33 indique la présence d'un module Trivium associé au processeur. Ce dernier est utilisé en tant que PRNG afin de mettre en oeuvre la stratégie de rejeu avec comparaison où le nombre de rejeu est aléatoire décrite dans la section 14.2. Enfin, la figure 33 indique également, via la présence de rectangle rouge, les composants modifiés pour mettre en oeuvre les solutions proposées. Le décodeur du processeur a été modifié pour décoder la nouvelle instruction *rpl*. Le composant *FETCH* a été modifié pour gérer l'évolution du pointeur programme lors du rejeu d'instructions. De plus, il permet, optionnellement, de recharger l'instruction depuis la mémoire durant la phase de rejeu d'une instruction. Le module de protection placé en amont du registre *IR* est alors en capacité de détecter une incohérence entre l'instruction en mémoire et l'instruction stockée dans le registre *IR*. Enfin, le contrôleur du processeur est modifié pour assurer la gestion de rejeu matériel d'instructions. Lorsqu'une instruction *rpl* est décodée, le processeur stocke les valeurs de n et w dans des registres locaux. Durant l'exécution des instructions protégées, des compteurs sont utilisés pour dénombrer le nombre de rejeux et contrôler la position dans la fenêtre de rejeu. De plus, pour chaque module de protection, le contrôleur gère l'activation des registres internes aux modules et vérifie l'état des signaux indiquant le statut des comparaisons ou des votes réalisés afin de lever une alerte *fault-detected* si nécessaire.

14.4 Principaux résultats

Les solutions proposées ont été implémentées sur une cible FPGA Xilinx Zynq 7020 via l'outil Vivado V2018.3. Dans la suite de cette section, les résultats de surface après placement-routage et de performance à la fréquence maximale obtenue pour chaque version de processeur sont présentés. Enfin, les résultats d'une campagne de simulation d'injection de fautes sont utilisés pour évaluer et comparer les protections proposées.

Un large ensemble de versions de processeur a été évalué. Chaque version présente un paramétrage matériel et une combinaison de protections distincts. La liste des versions de processeurs les plus représentatives est présentée ci-dessous.

TABLE 15 – Résultats d'implémentation FPGA (Zynq 7020) des principales versions de processeur (V)

V	Surface						Fréquence		
	LUT	%1	%2	FF	%1	%2	MHz	%1	%2
base	666	+00	-35	297	+00	-57	315	+00	+02
base + PRNG	1022	+53	+00	688	+132	+00	310	-02	+00
dc	775	+16	-24	398	+34	-42	311	-01	+00
rrcu/rrcv	1136	+71	+11	789	+166	+15	308	-02	-01
tv	821	+23	-20	465	+57	-32	312	-01	+01
dc+tv+rrcu+rrcv	1245	+87	+22	863	+191	+25	303	-04	-02

— "base" : processeur original

— "base+PRNG" : processeur original intégrant le PRNG Trivium ;

— "sw dc" : processeur *base* exécutant un code applicatif protégé en utilisant une solution logicielle pour la stratégie de duplication avec comparaison

— "dc" : processeur implémentant notre stratégie de rejeu matériel d'instruction avec comparaison pour $n = 1$ (à comparer à la version "sw dc")

— "rrcu" : processeur implémentant notre stratégie de rejeu matériel d'instruction avec comparaison où n est aléatoire et uniforme pour toutes les instructions de la fenêtre d'instructions protégées

— "rrcv" : processeur implémentant notre stratégie de rejeu matériel d'instruction avec comparaison où n est aléatoire et variable pour chaque instruction de la fenêtre d'instructions protégées

— "sw tv" : processeur *base* exécutant un code applicatif protégé en utilisant une solution logicielle pour la stratégie de triplication avec vote majoritaire

— "tv" : processeur implémentant notre stratégie de triplication avec vote majoritaire (à comparer avec "sw tv") ;

La table 15 présente les résultats d'implémentation des principales versions de processeur présentées ci-dessus. Les versions "sw dc" et "sw tv" ne sont pas présentées puisqu'elles sont identiques à la version "base" (seul le logiciel exécuté diffère). Pour chaque métrique, la colonne "%1" indique la différence relative, en pourcent, avec la version "base". La colonne "%2" indique la différence relative, en pourcent, avec la version "base+PRNG". Ces résultats montrent que les protections proposées ne pénalisent pas la fréquence de fonctionnement maximale du processeur. L'intégration de la stratégie de rejeu matériel d'instruction avec comparaison (dc) engendre un surcoût de 16% en LUT et 34% en Flip-Flop (FF) par rapport à la version "base" du processeur. L'intégration de la stratégie de triplication avec vote majoritaire (tv) engendre un surcoût de 23% en LUT et 57% en Flip-Flop par rapport à la version "base" du processeur. Enfin, l'intégration de la stratégie de rejeu matériel d'instruction avec comparaison où n est aléatoire engendre un surcoût de 11% en LUT et 15% en Flip-Flop par rapport à la version "base+PRNG" du processeur. La dernière ligne de la table 15 présente les résultats d'implémentation d'une version de processeur capable d'exécuter des applications pouvant faire appel à tous les mécanismes de protection proposés. Il est intéressant de noter que cette version complète engendre un surcoût de 22% en LUT et 25% en Flip-Flop par rapport à la version "base+PRNG" du processeur.

La table 16 présente une synthèse des performances temporelles et des tailles de codes pour un ensemble de quatre fonctions largement utilisées au sein d'applications de sécurité : *verify_pin*, *atoi*, *memcpy* et *trivium*. Dans cette table *CT* représente la moyenne des temps d'exécution (en *ns*) des quatre applications, *NC* représente la moyenne du nombre de cycles d'horloge nécessaire

TABLE 16 – Performances temporelles moyennes et tailles de code pour différentes versions de processeur

V	CT	NC	CS	RCT	RNC	RCS
base	422.3	132.8	33.8	1.0	1.0	1.0
sw dc	986.9	310.2	95.0	2.3	2.3	2.8
dc	875.0	271.8	39.8	2.1	2.0	1.2
rrcu	1479.2	455.0	39.8	3.5	3.4	1.2
rrcv	1452.4	446.8	39.8	3.4	3.4	1.2
sw tv	3872.1	1217.2	364.8	9.2	9.2	10.8
tv	1228.2	382.2	39.8	2.9	2.9	1.2

TABLE 17 – Synthèse des résultats issus des simulations de fautes

V	IS	DCF	EOK	DHW	DSW	DEC
base	20.3	15.7	49.6	0.0	0.0	14.4
sw dc	2.0	15.1	40.1	0.0	29.9	12.9
dc	0.2	4.8	44.6	46.8	0.0	3.5
rrcu	0.4	3.6	39.5	54.0	0.0	2.5
rrcv	0.2	3.9	38.0	55.2	0.0	2.7
sw tv	2.8	15.0	67.2	0.0	2.2	12.7
tv	0.6	6.7	81.3	6.8	0.0	4.6

à l'exécution de chacune des quatre applications et *CS* représente la moyenne des tailles de code (en nombre d'instructions) de chacune des quatre applications. Les trois dernières colonnes de la table 16 présentent les valeurs précédentes relativement à la version de processeur "base".

Les résultats présentés dans la table 16 montrent que les solutions proposées permettent de réduire significativement l'impact des protections sur la taille du code. De plus, contrairement aux approches purement logicielles, les solutions proposées permettent de limiter l'impact sur les performances temporelles au nombres de rejeux (c.-à-d. environ $\times 2$ pour la duplication avec comparaison et environ $\times 3$ pour la triplication avec vote majoritaire).

Dans le but d'évaluer le niveau de sécurité offert par les solutions proposées, une campagne de simulation d'injection de fautes a été réalisée pour l'ensemble des applications précédemment citées. Pour chaque simulation réalisée, un registre cible, un cycle d'injection et un type de faute (collage à 0, collage à 1 ou bit flip à une position aléatoire dans le registre cible) sont choisis aléatoirement selon une distribution uniforme. La table 17 présente une synthèse des résultats moyennés obtenus lors d'une campagne de simulations où une seule faute est injectée par simulation. Six métriques sont utilisées pour analyser les résultats obtenus : *IS* correspond au pourcentage d'injections ayant mené à une modification de l'état du processeur (registres et mémoire) après une fin légitime de l'application exécutée. Ce cas est jugé favorable à l'attaquant. *DCF* correspond au pourcentage d'injections ayant mené à une divergence du flot de contrôle. *EOK* correspond au pourcentage d'injections n'ayant pas impacté l'état du processeur (registres et mémoire) après une fin légitime de l'application exécutée. *DSW* correspond au pourcentage d'injections ayant mené à une détection par un mécanismes logiciel de duplication avec comparaison ou de triplication avec vote majoritaire. *DHW* correspond au pourcentage d'injections ayant mené à une détection par une stratégie de rejeu matériel d'instruction. *DEC* correspond

au pourcentage d'injections ayant mené à une erreur de décodage d'une instruction. Les résultats présentés dans la table 17 montrent que les solutions proposées fournissent une protection légèrement plus performante que les approches logicielles comparables ($IS<1\%$). De plus, on remarque que les solutions proposées réduisent le pourcentage d'injections menant à une divergence du flot de contrôle. Enfin, il est intéressant de remarquer que la stratégie de triplication avec vote majoritaire (tv) proposée permet d'améliorer sensiblement le pourcentage d'injections n'ayant pas impacté l'état du processeur (EOK) comparée à une approche équivalente purement logicielle ($sw\ tv$).

Les résultats obtenus durant ces travaux permettent de confirmer l'intérêt d'étudier des mécanismes de protections matériels intégrés aux processeurs permettant de se substituer ou de compléter les protections logicielles existantes. Un sous-ensemble des contributions présentées dans cette section a été présenté à la conférence DDECS en 2022 [Ait Manssour 2022]. Un article journal est également en préparation.

15 Protection d'un processeur avec DIFT contre des attaques physiques

Depuis octobre 2021, un travail de thèse s'intéresse à la robustesse des mécanismes de DIFT matériel face à des attaques physiques. Dans ces travaux, nous nous intéressons particulièrement au mécanisme de DIFT dit *in-core*. C'est à dire que les analyses DIFT sont réalisées au sein du coeur du processeur via une extension des étages de pipeline et du jeu d'instructions. Ces extensions n'ayant pas été développées en considérant les attaques physiques, elles doivent être repensées afin d'assurer l'efficacité du mécanisme de protection DIFT en présence, par exemple, de perturbations. Bien qu'au moment de l'écriture de ce manuscrit l'étude et le développement des premières contributions soient en cours, ce projet illustre la direction que je souhaite donner à l'axe de recherche "sécurité des processeurs embarqués".

15.1 Contexte des travaux

Ce travail de recherche réalisé dans le cadre de la thèse de M. William Pensec, s'appuie sur l'architecture DIFT *in-core* proposée dans [Palmiero 2018] et publiquement disponible [13]. La figure 34 présente cette architecture, nommée D-RI5CY, basée sur le processeur RISC-V RI5CY [14].

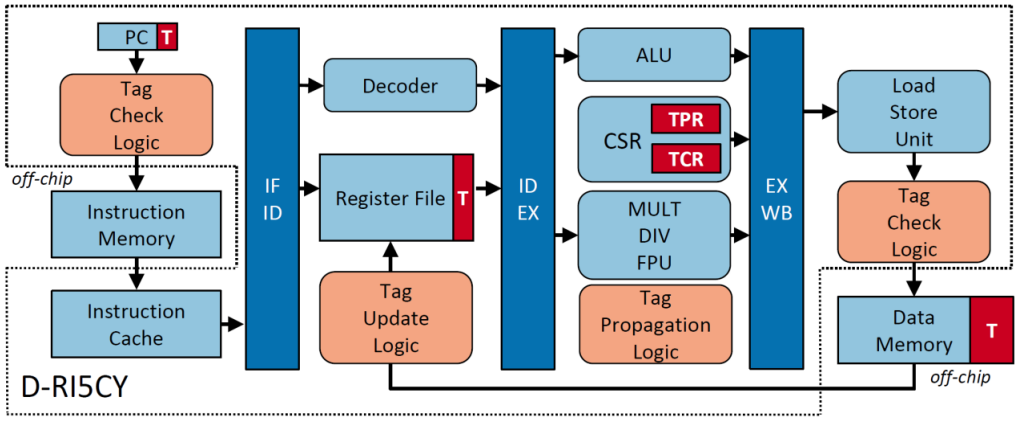


FIGURE 34 – Vue d'ensemble du processeur D-RI5CY (extrait de [Palmiero 2018])

La figure 34 met en lumière les modifications apportées au processeur d'origine. Tout d'abord, la mémoire de données, le banc de registre et le registre de pointeur programme ont été étendus pour stocker les tags associés aux données manipulées. Le rôle de l'unité *Tag Update Logic* est de mettre à jour les tags au sein du banc de registre. Cela est réalisé lors d'une lecture mémoire ou suite à la réalisation d'une opération arithmétique. L'unité *Tag Propagation Logic* est responsable de la mise à jour des tags. Cette opération est réalisée en parallèle des opérations arithmétiques et répond à une politique de propagation définie via le registre de configuration TPR (*Tag Propagation Register*). Enfin, les deux unités *Tag Check Logic* ont la charge d'appliquer la politique de sécurité définie via le registre TCR (*Tag Check Register*). En cas de violation de la politique de sécurité, une exception est levée.

15.2 Objectifs des travaux

Le processeur D-RI5CY permet de mettre en place une politique de sécurité permettant de se prémunir contre différentes attaques logicielles (p.ex. les attaques basées sur un débordement de

13. https://github.com/sld-columbia/riscv-dift
14. https://www.pulp-platform.org

tampons). Cependant, les éléments matériels nécessaires au bon fonctionnement du mécanisme de protection DIFT n'ont pas été développés en considérant un modèle de menace intégrant des attaques physiques.

Par conséquent, les travaux de thèse consistent à concevoir un nouveau mécanisme de DIFT offrant, en plus d'une protection face à des attaques logicielles, de la robustesse contre certaines attaques physiques (p. ex. analyse de la consommation de puissance et perturbation électromagnétique). Pour cela, nous combinerons et intégrerons à la micro-architecture du processeur une large gamme de méthodes de protection afin de les évaluer de façon systématique : masquage, obscurcissement, ajout d'aléa pour augmenter la robustesse à des attaques par observation et redondance spatiale/temporelle, détection et correction d'erreurs pour augmenter la robustesse à des attaques par perturbation. Nous analyserons les meilleures combinaisons et configurations des protections existantes et proposerons de nouvelles protections. Parmi les nouvelles contre-mesures que nous souhaitons étudier, la question de la propagation des tags de façon sécurisée nous paraît centrale, aussi nous explorerons la mise en oeuvre d'étiquettes encodées (pour la détection d'erreurs) et masquées (pour la réduction des fuites par canaux auxiliaires). Une telle approche nécessite une arithmétique dédiée que nous étudierons également.

Dans le cadre de ces travaux, les solutions proposées seront purement matérielles. Par conséquent, aucune modification de la chaîne de compilation n'est envisagée (sauf potentiellement l'ajout d'instructions de configuration). Les mécanismes de protections proposés seront intégrés à un processeur D-RI5CY, implémenté sur circuit FPGA afin de pouvoir prototyper et évaluer en terme de performance, de consommation d'énergie, de surface de circuit et de sécurité les solutions proposées.

À la date de rédaction de ce document, les solutions envisagées ne sont pas encore suffisamment matures pour être présentées plus en détail.

16 Conclusion

La conception de processeurs embarqués nécessite de prendre en compte un modèle de menace large intégrant les attaques logiques et physiques tout en respectant des contraintes fortes en terme de performance temporelle, de surface et de consommation d'énergie. Les travaux réalisés depuis 2015 ont permis de proposer des mécanismes de protections originaux pouvant participer à la conception de systèmes embarqués de confiance.

Les travaux menés dans le cadre du projet HardBlare entre 2015 et 2019 ont montré que la co-conception logicielle/matérielle peut permettre une mise en oeuvre efficace au sein d'un système embarqué d'un mécanisme de DIFT dont le surcoût aurait pu être jugé trop important pour une application dans le monde de l'embarqué. La preuve de concept réalisée démontre la pertinence d'une approche de la sécurité basé sur une collaboration forte entre la couche logicielle, permettant un management flexible de la sécurité, et la couche matérielle fournissant des services de sécurité dédiés efficaces temporellement et énergétiquement.

Les travaux débuté en 2019 dans le cadre des travaux de thèse de Mme Noura Ait Manssour adressent la problématique des attaques physiques par perturbation au sein des processeurs embarqués. Ces travaux ont permis de proposer une solution basée sur le rejeu matériel d'instructions permettant d'égaler les niveaux de sécurité offerts par des protections logicielles équivalentes tout en réduisant significativement l'impact de la protection sur les performances temporelles d'une part et sur la taille des codes à protéger d'autre part.

Enfin, depuis 2021, les travaux de thèse de M. William Pensec se focalisent sur la conception, la mise en oeuvre et l'évaluation d'un mécanisme de DIFT *in-core* en considérant un modèle de menace intégrant des attaques logiques et des attaques physiques. Ce travail s'intègre dans une volonté de concevoir des processeurs embarqués devant faire face à des menaces multiples.

La table 18 fournie quelques éléments de bilan des activités de cet axe de recherche.

Table 18 – bilan des activités pour l'axe de recherche "sécurité des processeurs embarqués"

Valorisation	5 communications internationales 2 communications nationales 1 communication par affiche
Collaborateurs	G. Gogniat (UBS), A. Tisserand (CNRS)), G. Hiet (Centrale Supelec), P. Cotret (ENSTA Bretagne),
Doctorants	M. A. Wahab, N. Ait Manssour, W. Pensec
Post-doc	A. K. Biswas
Financeurs	Région Bretagne, LabEx CominLabs

16.1 Liste des publications associées à l'axe

16.1.1 Communications internationales avec comité de lecture et actes

[AIT 2022] N. Ait Manssour, V. Lapotre, G. Gogniat, A. Tisserand, **Processor Extensions for Hardware Instruction Replay against Fault Injection Attacks**, in Proc. of 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2022.

[WAH 2018a] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, A. Kumar Beswas, G. Gogniat, V. Lapotre, **A novel lightweight hardware-assisted static instrumentation approach for ARM SoC using debug components**, in proc of Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2018.

[WAH 2018b] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, A. Kumar Beswas, G. Gogniat and V. Lapotre, **A small and adaptive coprocessor for information flow tracking in ARM SoCs**, in proc of 2018 International Conference on Reconfigurable Computing and FPGAs (Reconfig), 2018.

[WAH 2017a] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre and G. Gogniat, **ARMHEx : A hardware extension for DIFT on ARM-based SoCs**, in Proc. of 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.

[WAH 2016a] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre and G. Gogniat, "**Towards a hardware-assisted information flow tracking ecosystem for ARM processors**, in Proc. of 26th International Conference on Field Programmable Logic and Applications (FPL), 2016.

16.1.2 Communication nationale avec comité de lecture et actes

[WAH 2017b] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre, G. Gogniat, **ARMHEx : a hardware extension for information flow tracking on ARM-based platforms**, Les Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI 2017), 2017.

16.1.3 Communications nationales invitées sans actes

[LAP 2019a] V. Lapotre, **A Hardware/software co-design approach for security analysis of application behavior - Applications on Dynamic Information Flow Tracking** , Journée "Nouvelles Avancées en Sécurité des Systèmes d'Information", Toulouse, 2019.

16.1.4 Communication par affiche

[WAH 2017c] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre, G. Gogniat, **ARMHEx : a framework for efficient DIFT in real-world SoCs**, 27th International Conference on Field Programmable Logic and Applications (FPL), 2017.

References

[Agoyan 2010] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson et Assia Tria. *When Clocks Fail : On Critical Paths and Clock Faults.* In Proc. International Conference on Smart Card Research and Advanced Applications (CARDIS), pages 182–193. Springer, Avril 2010.

[Ait Manssour 2022] Noura Ait Manssour, Vianney Lapotre, Gogniat Guy et Arnaud Tisserand. *Processor Extensions for Hardware Instruction Replay against Fault Injection Attacks.* In DDECS : 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, Prague, Czech Republic, Avril 2022. IEEE.

[Bar-El 2006] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall et Claire Whelan. *The Sorcerer's Apprentice Guide to Fault Attacks.* Proceedings of the IEEE, vol. 94, no. 2, pages 370–382, F ?rier 2006.

[Barenghi 2010] Alessandro Barenghi, Luca Breveglieri, Israel Koren, Gerardo Pelosi et Francesco Regazzoni. *Countermeasures Against Fault Attacks on Software Implemented AES : Effectiveness and Cost.* In Proc. Workshop on Embedded Systems Security (WESS), pages 7 :1–10, Scottsdale, AZ, USA, Octobre 2010. ACM.

[Boneh 1997] Dan Boneh, Richard A. DeMillo et Richard J. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults.* In Proc. Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT), volume 1233 of *LNCS*, pages 37–51. Springer, 1997.

[Boneh 2001] Dan Boneh, Richard A. DeMillo et Richard J. Lipton. *On the Importance of Eliminating Errors in Cryptographic Computations.* Journal of Cryptology, vol. 14, no. 2, pages 101–119, Mars 2001.

[Bozzato 2019] Claudio Bozzato, Riccardo Focardi et Francesco Palmarini. *Shaping the Glitch : Optimizing Voltage Fault Injection Attacks.* Transactions on Cryptographic Hardware and Embedded Systems (TCHES), pages 199–224, F ?rier 2019.

[Brant 2021] Christopher Brant, Prakash Shrestha, Benjamin Mixon-Baca, Kejun Chen, Said Varlioglu, Nelly Elsayed, Yier Jin, Jedidiah Crandall et Daniela Oliveira. *Challenges and Opportunities for Practical and Effective Dynamic Information Flow Tracking.* ACM Computing Surveys (CSUR), 2021.

[Chen 2008] Shimin Chen, Michael Kozuch, Theodoros Strigkos, Babak Falsafi, Phillip B. Gibbons, Todd C. Mowry, Vijaya Ramachandran, Olatunji Ruwase, Michael Ryan et Evangelos Vlachos. *Flexible Hardware Acceleration for Instruction-Grain Program Monitoring.* In 2008 International Symposium on Computer Architecture, pages 377–388, 2008.

[Colombier 2021] Brice Colombier, Lilian Bossuet, Paul Grandamme, Julien Vernay, Emilie Chanavat, Lucie Bon et Bruno Chassagne. *Multi-spot Laser Fault Injection Setup : New Possibilities for Fault Injection Attacks.* In 20th Smart Card Research and Advanced Application Conference - CARDIS 2021, Lübeck, Germany, Novembre 2021.

[Dalton 2007] Michael Dalton, Hari Kannan et Christos Kozyrakis. *Raksha : A Flexible Information Flow Architecture for Software Security.* SIGARCH Comput. Archit. News, vol. 35, no. 2, page 482–493, jun 2007.

[Dehbaoui 2012] Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, Philippe Orsatelli, Philippe Maurine et Assia Tria. *Injection of Transient Faults using Electromagnetic Pulses – Practical Results on a Cryptographic System.* IACR Cryptology ePrint Archive, Mars 2012.

[Dehbaoui 2013] Amine Dehbaoui, Amir-Pasha Mirbaha, Nicolas Moro, Jean-Max Dutertre et Assia Tria. *Electromagnetic Glitch on the AES Round Counter.* In Proc. Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE), pages 17–31. Springer, Mars 2013.

[Dhawan 2015] Udit Dhawan, Catalin Hritcu, Raphael Rubin, Nikos Vasilakis, Silviu Chiricescu, Jonathan M. Smith, Thomas F. Knight, Benjamin C. Pierce et Andre DeHon. *Architectural Support for Software-Defined Metadata Processing.* In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, page 487–502, New York, NY, USA, 2015. Association for Computing Machinery.

[Dureuil 2015] Louis Dureuil, Marie-Laure Potet, Philippe Choudens, Cécile Dumas et Jessy Clédière. *From Code Review to Fault Injection Attacks : Filling the Gap using Fault Model Inference.* In Proc. International Conference on Smart Card Research and Advanced Applications (CARDIS), pages 107–124. ACM, Novembre 2015.

[Dureuil 2016] Louis Dureuil, Guillaume Petiot, Marie-Laure Potet, Thanh-Ha Le, Aude Crohen et Philippe de Choudens. *FISSC : A Fault Injection and Simulation Secure Collection.* In Proc. International Conference on Computer Safety, Reliability, and Security (SAFECOMP), pages 3–11. Springer, Septembre 2016.

[Efstathopoulos 2005] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek et Robert Morris. *Labels and Event Processes in the Asbestos Operating System.* SIGOPS Oper. Syst. Rev., vol. 39, no. 5, page 17–30, oct 2005.

[Georget 2017] Laurent Georget, Mathieu Jaume, Guillaume Piolle, Frédéric Tronel et Valérie Viet Triem Tong. *Information Flow Tracking for Linux Handling Concurrent System Calls and Shared Memory.* In Alessandro Cimatti et Marjan Sirjani, editeurs, 15th International Conference on Software Engineering and Formal Methods (SEFM 2017), LNCS, pages 1–16, Trento, Italy, Septembre 2017. Springer International Publishing.

[Guthaus 2001] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge et R. B. Brown. *MiBench : A Free, Commercially Representative Embedded Benchmark Suite.* In Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop, WWC '01, page 3–14, USA, 2001. IEEE Computer Society.

[Kannan 2009] Hari Kannan, Michael Dalton et Christos Kozyrakis. *Decoupling Dynamic Information Flow Tracking with a dedicated coprocessor.* In 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, pages 105–114, 2009.

[Karaklajić 2013] Duško Karaklajić, Jörn-Marc Schmidt et Ingrid Verbauwhede. *Hardware Designer's Guide to Fault Attacks.* IEEE Transactions on Very Large Scale Integration Systems, vol. 21, no. 12, pages 2295–2306, F ?rier 2013.

[Moro 2014] Nicolas Moro, Karine Heydemann, Emmanuelle Encrenaz et Bruno Robisson. *Formal Verification of a Software Countermeasure Against Instruction Skip Attacks.* Journal of Cryptographic Engineering (JCEN), vol. 4, no. 3, pages 145–156, F ?rier 2014.

[Nasr Allah 2020] Mounir Nasr Allah. *Contrôle de flux d'information par utilisation conjointe d'analyse statique et dynamique accélérée matériellement.* Theses, CentraleSupélec, D ?embre 2020.

[Newsome 2005] James Newsome et Dawn Xiaodong Song. *Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software.* In Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS 2005), volume 5, pages 3–4, 2005.

[Palmiero 2018] Christian Palmiero, Giuseppe Di Guglielmo, Luciano Lavagno et Luca P. Carloni. *Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications.* In Proc. IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.

[Qin 2006] Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou et Youfeng Wu. *LIFT : A Low-Overhead Practical Information Flow Tracking System for Detecting Se-*

curity Attacks. In 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), pages 135–148, 2006.

[Quisquater 2002] Jean-Jacques Quisquater et David Samyde. *Eddy Current for Magnetic Analysis with Active Sensor.* Proc. eSMART, pages 185–194, Septembre 2002.

[Roy 2009] Indrajit Roy, Donald E. Porter, Michael D. Bond, Kathryn S. McKinley et Emmett Witchel. *Laminar : Practical Fine-Grained Decentralized Information Flow Control.* SIGPLAN Not., vol. 44, no. 6, page 63–74, jun 2009.

[Ruwase 2008] Olatunji Ruwase, Phillip B. Gibbons, Todd C. Mowry, Vijaya Ramachandran, Shimin Chen, Michael Kozuch et Michael Ryan. *Parallelizing Dynamic Information Flow Tracking.* In Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, SPAA '08, page 35–45, New York, NY, USA, 2008. Association for Computing Machinery.

[Skorobogatov 2003] Sergei P. Skorobogatov et Ross J. Anderson. *Optical Fault Induction Attacks.* In Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pages 2–12. Springer, F ?rier 2003.

[Suh 2004] G. Edward Suh, Jae W. Lee, David Zhang et Srinivas Devadas. *Secure Program Execution via Dynamic Information Flow Tracking.* SIGARCH Comput. Archit. News, vol. 32, no. 5, page 85–96, oct 2004.

[Thati 2019] Venu Babu Thati, Jens Vankeirsbilck, Davy Pissoort et Jeroen Boydens. *Selective Duplication and Selective Comparison for Data Flow Error Detection.* In Proc. International Conference on System Reliability and Safety (ICSRS), pages 10–15. IEEE, Novembre 2019.

[Timmers 2016] Niek Timmers et Albert Spruyt. *Bypassing Secure Boot using Fault Injection.* Black Hat Europe, F ?rier 2016.

[Trouchkine 2021] Thomas Trouchkine, Sébanjila Kevin K Bukasa, Mathieu Escouteloup, Ronan Lashermes et Guillaume Bouffard. *Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models.* Journal of Cryptographic Engineering, vol. 11, no. 4, pages 353–367, Novembre 2021.

[Vasselle 2017] Aurélien Vasselle, Hughes Thiebeauld, Quentin Maouhoub, Adèle Morisset et Sébastien. Ermeneux. *Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot.* In Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pages 41–48, Santa Barbara, CA, USA, Ao ?t 2017. IEEE.

[Venkataramani 2008] Guru Venkataramani, Ioannis Doudalis, Yan Solihin et Milos Prvulovic. *FlexiTaint : A programmable accelerator for dynamic taint propagation.* In 2008 IEEE 14th International Symposium on High Performance Computer Architecture, pages 173–184, 2008.

[Wahab 2018a] Muhammad Abdul Wahab. *Hardware support for the security analysis of embedded softwares : applications on information flow control and malware analysis.* Theses, CentraleSupélec, D ?embre 2018.

[Wahab 2018b] Muhammad Abdul Wahab, Pascal Cotret, Mounir Nasr Allah, Guillaume Hiet, Arnab Kumar Biswas, Vianney LapÔtre et Guy Gogniat. *A small and adaptive coprocessor for information flow tracking in ARM SoCs.* In 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pages 1–8, 2018.

[Zeldovich 2008] Nickolai Zeldovich. Securing untrustworthy software using information flow control. PhD Thesis Stanford University, 2008.

Cinquième partie

Conclusion et perspectives

Ce document présente une synthèse de mes travaux de recherche depuis ma nomination en tant que Maître de Conférences au sein de l'Université Bretagne Sud en septembre 2014. Entre 2014 et 2020, j'ai mené mes travaux au sein de l'équipe Méthodes et Outils pour les Circuits et Systèmes (MOCS) du laboratoire Lab-STICC. Puis, suite à une restructuration du laboratoire en 2020, j'ai intégré l'équipe architectures matérielles et outils de CAO (ARCAD) dont je suis aujourd'hui le responsable. Mes travaux se positionnent dans le domaine de la conception des systèmes numériques de confiance. La synthèse de mes travaux a été présenté selon trois axes de recherche. Le premier concerne l'implémentation de fonctions cryptographiques. Le second correspond à l'étude de protections face aux attaques exploitant la microarchitecture. Le troisième se focalise sur la sécurité des processeurs embarqués. Dans les années à venir, je souhaite poursuivre mes efforts dans le domaine de la conception de systèmes sur puce de confiance. Dans la suite, je propose un ensemble de perspectives à court, moyen et long terme.

À court terme, je projette d'enrichir mes travaux dans les axes concernant la protection face aux attaques exploitant la microarchitecture et la sécurité des processeurs embarqués. Le projet SCRATCHS, actuellement en cours participe à ma volonté de renforcer la collaboration entre les couches logicielles et les couches matérielles dans le but de développer des systèmes répondant à des contraintes de sécurité élevées. Ce projet s'inscrit parfaitement dans ma vision de la sécurité où la conception des éléments de la chaîne de confiance se doivent d'être étudiés, développés et évalués conjointement.

Le projet de thèse de M. William Pensec, également actuellement en cours, participe à cette ambition à travers la prise en compte d'un modèle de menace élargi lors de la conception de processeurs embarqués. En effet, ces travaux proposent de questionner l'efficacité d'un mécanisme DIFT, assurant une protection contre des attaques logicielles, en présence d'attaques physiques. Cette thématique s'est renforcée fin 2022 par le démarrage des travaux de thèses de M. Hongwei Zhao et du travail post-doctoral de M. Kamel Aizi. Le premier s'intéressera à l'impact des attaques physiques au sein d'un système sur puce conçu pour devenir une racine de confiance au sein d'un système complexe (p.ex. le système sur puce OpenTitan [15]). En particulier, il s'agit également d'étudier comment l'information se propage dans l'architecture de communication en cas d'attaque. Ce sujet vise donc à mettre en place une analyse des vulnérabilités, d'étudier la propagation des attaques et de proposer des contremesures de façon à développer des architectures de communication résistantes aux attaques physiques. Le second, étudiera l'impact de l'intégration de moniteurs de sécurité au sein de processeurs embarqués sur les fuites d'information observables via des canaux auxiliaires physiques tel que la consommation de puissance ou le rayonnement électromagnétique. En effet, pour effectuer des contrôles de sécurité, ces moniteurs doivent extraire et manipuler des informations internes du processeur (compteur de programme, instruction, adresses et données...). Ainsi, lors de l'exécution d'une application sensible tels que les algorithmes de cryptographie, les protocoles d'authentification, la gestion du contrôle d'accès ou des fonctions à haut privilège du système d'exploitation, un moniteur de sécurité manipule des informations sensibles. Les traitements effectués par le moniteur de sécurité sur ces informations pourraient devenir une nouvelle source de fuite d'informations grâce à l'analyse physique de canaux auxiliaires.

Les travaux de Mme Noura Ait Manssour, qui se focalisent sur l'étude et la mise en oeuvre de protections contre des attaques physiques par perturbation, seront également poursuivi dans le cadre du projet ARSENE du PEPR Cybersécurité. Ces futurs travaux s'intéresseront particulièrement à l'intégration et la mise en oeuvre de combinaisons de protections au sein d'un

15. https://opentitan.org/

processeur RISC-V et l'évaluation de l'approche face à des attaques par observation et par perturbation. De plus, il s'agira d'évaluer l'impact des optimisations de la microarchitecture telle que la profondeur du pipeline, la prédiction de branchement ou la spéculation sur l'efficacité des solutions proposées.

À moyen terme, le développement de protections s'appuyant sur une co-conception logicielle/matérielle est, je pense, une approche particulièrement prometteuse dans le but de proposer des protections contre des menaces complexes. En effet, les couches logicielles n'ont généralement que peu de détails concernant le comportement réel de l'architecture matérielle sur laquelle elles s'exécutent. Du côté de la couche matérielle, les interactions entre les composants logiciels et les divers besoins applicatifs sont généralement méconnus. Cependant, pour assurer un haut niveau de sécurité, je pense qu'il est nécessaire de construire des mécanismes de protections où l'échange d'information entre ces couches est régulier. Bien que cette approche ait déjà été appliquée pendant le projet HardBlare présenté dans ce document et soit poursuivie dans le cadre du projet SCRATCHS actuellement en cours, des travaux futurs devront adresser de nouveaux challenges.

Parmi ceux-ci, nous pouvons évoquer l'intégration de méthodes de modélisation et de vérification formelle dès la conception des composants matériels participant à la mise en oeuvre d'un mécanisme de protection dans le but de consolider, voir prouver, l'efficacité du mécanisme proposé. Un point d'investigation supplémentaire me semble être le développement de méthodologies permettant l'étude du partitionnement des traitements entre les couches logicielles et les couches matérielles lors de la conception d'un mécanisme de protection au sein de systèmes sur puce complexes. Si des méthodes de co-conception ont été proposées par le passé pour des objectifs d'optimisation des performances temporelles, de la consommation d'énergie et du coût silicium, la dimension sécurité doit également être explorée. Par exemple, l'intégration d'un traitement au sein d'un bloc matériel dédié permettra de le protéger de certaines menaces logicielles. Cependant, il sera généralement plus difficile de le faire évoluer dans le temps afin de s'adapter à de nouvelles menaces. Il me semble donc important d'étudier des approches de co-conception s'appuyant sur des métriques nouvelles permettant d'explorer les compromis possibles tout en garantissant le juste niveau de sécurité et de performance.

La conception de systèmes embarqués de confiance nécessite également de prendre en compte leur intégration dans un environnement de plus en plus connecté. Dans ce contexte, il est nécessaire de construire des solutions de sécurité pouvant répondre à des menaces complexes pouvant s'appuyer sur une combinaison d'attaques physiques, logicielles et réseaux. La prise en compte de ces menaces nécessite, de mon point de vue, une approche pluridisciplinaire permettant la modélisation de la menace, la mise en oeuvre d'un ensemble de protections assurant une défense en profondeur du système, la surveillance en temps réel du système permettant la détection et la réaction face à des comportements malveillants et enfin le maintien en condition de sécurité via la mise en oeuvre de mises à jour de sécurité régulières. J'envisage particulièrement de mener des travaux permettant l'étude et le développement de solutions d'analyse de sécurité embarquées au sein de systèmes sur puces permettant une surveillance globale du système dans le but de détecter et de réagir face à des comportements malveillants du système. Ces solutions pourront s'appuyer sur un ensemble d'information récolté en temps réel sur l'ensemble du système sur puce (transaction sur les interconnexions, compteurs de performance, traces d'exécution, consommation d'énergie, motif d'accès à la mémoire,...) et sur la mise en oeuvre de modèles d'apprentissage.

À long terme, je souhaite interroger le modèle habituel du couple système d'exploitation / systèmes sur puce en mettant les objectifs de sécurité, de performance, d'énergie et de coût au même niveau d'importance. On observe par exemple que la mise en oeuvre de *Trusted Execution Environment* (TEE) favorisant l'isolation entre les processus afin de garantir des propriétés de confidentialité et d'intégrité s'articulent aujourd'hui sur des architectures matérielles qui ont été à l'origine conçues pour maximiser la réutilisation et le partage de ressources. Il me semble alors nécessaire de mener une réflexion dans le but d'identifier de nouvelles architectures matérielles

(et les couches logicielles associées) permettant, par construction, de garantir un ensemble de propriétés de sécurité. Cela pourrait, par exemple, passer par la mise en oeuvre d'éléments matériels dédiés, participant à la gestion des ressources et des services de sécurité, totalement isolés des ressources mises en oeuvre pour réaliser les traitements applicatifs. Cette approche pourrait reposer sur une co-conception logicielle/matérielle des fonctions critiques prises en charges aujourd'hui par un hyperviseur ou un système d'exploitation. Une telle approche nécessiterait alors de repenser conjointement l'architecture des systèmes sur puce et la pile logicielle dédiée à la gestion des ressources matérielles.

Un autre élément de réflexion à long terme concerne la mise en oeuvre des technologies nouvelles au service de la sécurité. On peut par exemple évoquer l'arrivée possible de l'ordinateur quantique et son application pour des fonctions de sécurité. Il serait alors intéressant d'étudier sa mise en oeuvre dans des systèmes hybrides, où composant "classiques" et "quantiques" collaborent, pour la réalisation de tâches liées à la détection et l'interruption d'attaques physiques ou logicielles en se reposant sur des capacités d'analyse des signaux faibles en temps réels.

COLLEGE MATHSTIC
DOCTORAL BRETAGNE
BRETAGNE OCEANE

Université Bretagne Sud

Titre : Contributions à la sécurité des systèmes embarqués face aux attaques logiques et physiques

Mot clés : Sécurité matérielle, processeurs embarqués, systèmes embarqués, protections face à des attaques logiques et physiques

Résumé : Les systèmes embarqués se répandent massivement dans les infrastructures critiques (industrie 4.0, ville intelligente, transports...) participant à l'augmentation de la surface d'attaque globale. En effet, ce type de système, souvent mal protégé et mal maîtrisé par les équipes en charge de la cybersécurité des systèmes d'information, peut devenir un point d'entrée privilégié pour l'attaquant. En raison de la proximité des systèmes embarqués avec un potentiel attaquant, il est nécessaire de considérer un modèle de menace large intégrant les attaques logiques et physiques lors de leur conception.

Les travaux présentés dans ce manuscrit adressent différents axes de recherche. Dans un premier axe, nous proposons des solutions pour l'étude et l'implémentation matérielle de fonctions cryptographiques sous contraintes de performance temporelle, de surface, d'efficacité énergétique et de sécurité. Dans un second axe, nous nous interessons au développement de protections face aux attaques exploitant la microarchitecture. Plus particulièrement, les travaux menés se focalisent sur les attaques s'appuyant sur l'observation de variations temporelles lors des accès mémoires durant l'exécution de codes logiciels pour extraire des informations sensibles manipulées par un programme victime. Les travaux réalisés proposent des solutions s'appuyant sur une collaboration logicielle / matérielle pour la détections de ces attaques, l'isolation des traitements sensibles ou la suppression du canal auxiliaire utilisé. Enfin, dans un troisième axe, nous étudions des architectures de processeurs embarqués intégrant, à la conception, des protections contre les attaques logiques et physiques. Ces travaux s'appuient fortement sur des éléments matériels afin de minimiser le coût lié à la sécurité en terme de performance et d'énergie.

Title: Contributions to the security of embedded systems against logical and physical attacks

Keywords: Hardware security, embedded processor, embedded systems, protection against logical and physical attacks

Abstract: Embedded systems are spreading massively in critical infrastructures (industry 4.0, smart city, transportation...), contributing to the increase of the global attack surface. Indeed, this type of system, which is often poorly protected and poorly controlled by the teams in charge of the cybersecurity, can become a privileged entry point for attackers. Due to the proximity of embedded systems to a potential attacker, it is necessary to consider a broad threat model integrating logical and physical attacks when designing such systems.

The work presented in this manuscript addresses several research axes. In a first axis, we propose solutions to study and implement cryptographic primitives under performance, area, energy and security constraints. In a second axis, we study and develop protections against microarchitectural attacks. More specifically, the proposed contributions focus on attacks based on the observation of temporal variations during memory accesses in order to extract sensitive information manipulated by a victim program. We propose solutions based on software / hardware cooperation to detect these attacks, isolate sensitive processes and mitigate such side-channel leakages. Finally, in a third axis, we study embedded processor architectures integrating, at the design stage, protections against both logical and physical attacks. This work relies heavily on hardware components in order to minimize the cost of security in terms of performances and energy.