Sécurité Haut-débit pour les Systèmes Embarqués à base de FPGAs

Jérémie Crenne

THESE / UNIVERSITE DE BRETAGNE SUD
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE DE BRETAGNE SUD
Mention : STIC
Ecole doctorale SICMA

présentée par
Jérémie Crenne

Laboratoire d'accueil :
Lab-STICC Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance / Lorient

Sécurité Haut-débit pour les Systèmes Embarqués à base de FPGAs

Thèse soutenue le 9 Décembre 2011
devant le jury composé de :

Olivier Sentieys
Professeur des universités / *président* / Université de Rennes 1

Lilian Bossuet
Maître de conférences, HDR / *rapporteur* / Université Jean Monnet

Christophe Jégo
Professeur des universités / *rapporteur* / IPB/ENSEIRB-MATMECA

Russell Tessier
Professeur de universités / *examinateur* / Université du Massachusetts

Jean Philippe Diguet
Directeur de recherche / *examinateur* / Université de Bretagne Sud

Pierre Bomel
Docteur ingénieur de recherche / *examinateur* / Université de Bretagne Sud

Guy Gogniat
Directeur de thèse / Université de Bretagne Sud

Sécurité haut-débit pour les systèmes embarqués à base de FPGAs

Jérémie Crenne

9 janvier 2012

Table des matières

Table des figures

Liste des tableaux

A nos futurs pionniers.

Résumé

« [...] *Puis, l'on ferra des récepteurs de télévision bijoux, comme il y a des postes de TSF bijoux. Des postes de poches, grands comme une lampe électrique. Plus besoin d'acheter un journal, l'on se branchera sur l'émission d'information, ou sur l'éditorial politique, ou sur la chronique de mode, ou sur le compte rendu sportif. Voir même sur un problème de mots croisés. Et la rue présentera un singulier spectacle.* »

R. Barjavel, « La télévision, oeil de demain », 1947.

C'est ainsi que l'auteur de romans de science fiction et d'anticipation René Barjavel, avait prédit dés la fin des années 40 l'avènement de ce que nous connaissons sous le nom de smartphones. Drôle de scène, en effet, que de voir des individus déambuler dans les rues, les yeux rivés sur l'objet au creux de leur main.

Pour le meilleur et pour le pire, l'avènement de la mise en réseau à l'échelle mondiale a rendu les systèmes embarqués omniprésents dans notre quotidien. Désormais dans le nuage, le nombre d'information personnel en transit et les vitesses de transfert toujours plus importants, imposent une sécurité adéquate. Cependant, le coût en général associé est économiquement dissuasif.

Proposer des solutions de sécurité ad-hoc pour ces systèmes restreints en ressources, est le propos de nos travaux. S'appuyant sur des techniques à la fois anciennes et récentes, nous montrons que le couple embarqué-sécurité peut s'accorder, et éviter ainsi, une inévitable procédure de divorce.

Abstract

« [...] *Then, we will build TV and radio like jewelry. TV in pockets, big as flashlights. No need to buy newspapers, we will connect ourselves to news, political, fashion, or sports programs. Or even to a crossword puzzle solver. And the street will present a singular spectacle.* »

R. Barjavel, « The eye of tomorrow », 1947.

French Scifi writer René Barjavel predicts in the late 40's the advent of what we know as smartphones. Funny scene, in fact, to see people into walking the streets, eyes fixed on the object in the palm of their hand.

For better or for worse, the advent of networking has globally made embedded systems ubiquitous in our daily lives. Now in the cloud, the number of personal information in transit is huge and transfer speeds are still more important and require adequate security.

However, the associated cost is generally economically deterrent. Offering ad-hoc security approaches for systems with such limited resources is the purpose of our work. Based on old and new techniques, we show that the pair embedded-security can be tuned, and avoid, an inevitable divorce.

Publications

Ci-dessous est la liste des publications durant ma période d'étudiant chercheur, i.e. de 2008 à 2011. La Figure 1 montre l'historique chronologique de ces publications.

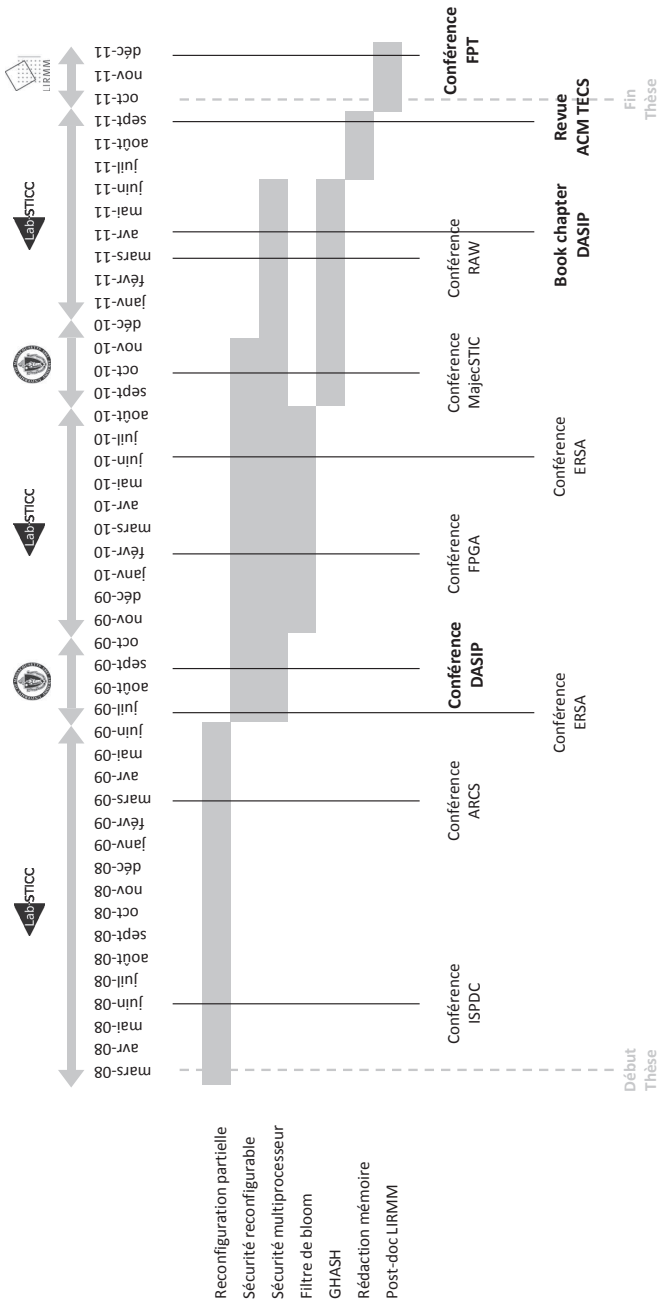

FIGURE 1 – Historique des publications durant la période de thèse

Journaux et Chapitre de Livre

J. Crenne, R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier and D. Unnikrishnan, **Configurable Memory Security in Embedded Systems**, *in ACM Transactions on Embedded Computer Systems (TECS)*, accepted/to appear.

J. Crenne, P. Bomel, G. Gogniat, J.-P Diguet, **End-to-End Bitstreams Repository Hierarchy for FPGA Partially Reconfigurable Systems**, *in Algorithm-Architecture Matching for Signal and Image Processing*, Springer, ISBN : 978-90-481-9964-8, pp. 171-194.

Conférences et Workshop Internationaux

J. Crenne, P. Cotret, G. Gogniat, R. Tessier, and J.-P. Diguet, **Efficient Key-Dependent Message Authentication in Reconfigurable Hardware**, *in the Proceedings of the International Conference on Field-Programmable Technology (FPT'11)*, December 12-14, 2011, New Delhi, India.

P. Cotret, J. Crenne, G. Gogniat, J.-P Diguet, L. Gaspar, G. Duc, **Distributed security for communications and memories in a multiprocessor architecture**, *in the Proceedings of the Reconfigurable Architecture Workshop (RAW'11)*, May 16-17, 2011 , Anchorage, Alaska, USA.

G. Gogniat, J. Vidal, L. Ye, J. Crenne, S. Guillet, F. de Lamotte, J.-P Diguet, P. Bomel, **Self-reconfigurable Embedded Systems : from Modeling to Implementation**, *in the Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'10)*, July 12-15, 2010, Las Vegas, Nevada, USA.

D. Unnikrishnan, R. Vadlamani, Y. Liao, A. Dwaraki, J. Crenne, L. Gao, R. Tessier, **Scalable Network Virtualization Using FPGAs**, *in the Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA'10)*, February 21-23, 2010, Monterey, California, USA.

J. Crenne, P. Bomel, G. Gogniat, J.-P Diguet, **UDP Partial Bitstreams Diffusion Through WLAN**, *in the Proceedings of the International Conference on Design and Architectures for Signal and Image Processing (DASIP'09)*, September 22-24, 2009, Sophia Antipolis, France.

J.-P Diguet, L. Ye, Y. Eustache, J. Crenne, P. Bomel, G. Gogniat, J. Vidal, F. de Lamotte, **Networked Self-adaptive Systems : An Opportunity for Configuring in the Large**, *in the Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'09)*, July 13-16, 2009, Las Vegas, Nevada, USA.

P. Bomel, J. Crenne, L. Ye, G. Gogniat, J.-P Diguet, **Ultra-Fast Downloading of Partial Bitstreams Through Ethernet**, *in the Proceedings of the International Conference on Architecture of Computing Systems (ARCS'09)*, March 10-13, 2009, Delft, The Netherlands.

P. Bomel, G. Gogniat, J.-P Diguet, J. Crenne, **Bitstreams Repository Hierarchy for FPGA Partially Reconfigurable Systems**, *in the Proceedings of the International Symposium on Parallel and Distributed Computing (ISPDC'08)*, July 1-5, 2008, Krakow, Poland.

Conférences Nationales

P. Cotret, J. Crenne, G. Gogniat, **Sécurisation des communications dans une architecture multi-processeurs**, *MAnifestation des JEunes Chercheurs en Sciences et Technologies de l'Information et de la Communication (MajecSTIC'10)*, October 14, 2010, Bordeaux, France.

Travaux en collaboration

Le Chapitre 3 a largement bénéficié de l'apport du Pr. Russell Tessier de l'Université du Massachusetts (UMASS) qui a significativement contribué à la présentation et à la robustesse des travaux.

Le Chapitre 4 est le fruit de la longue collaboration entre l'Université de Bretagne-Sud (UBS) et l'UMASS et des deux laboratoires porteurs, le Laboratoire en Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) et le Reconfigurable Computing Group (RCG). Ces travaux sont également redevables des apports du docteur Romain Vaslin à l'origine de l'approche. Bien que plus expérimentaux, les chapitres 6 et l'Annexe sont issus de cette même entente.

Le Chapitre 5 sur la reconfiguration dynamique partielle est fondé sur les travaux initiés par Pierre Bomel du Lab-STICC à l'UBS.

Introduction

Un réseau de portes est un circuit intégré qui consiste en une grille de transistors fabriquée sur un wafer de silicium. Différents arrangements des couches de métal pour les interconnexions peuvent être ajoutés pour définir la fonction du circuit. Ceci permet à la même masse produite de wafers d'être utilisée pour effectuer différentes fonctions logiques. Les réseaux de portes ont une intégration limitée mais sont moins onéreux à la fabrication que les circuits intégrés pour application spécifique (Application Specific Integrated Circuit, ASIC). La programmabilité est une propriété qui permet à la fonctionnalité d'un dispositif d'être modifié sur le champ, c'est à dire en dehors de l'usine. En ajoutant cette propriété les réseaux de portes nous donnent des réseaux de portes configurables sur le champ (Field-Programmable Gate Array, FPGA), des dispositifs semi-conducteur construits à partit de réseaux d'interconnexions et de blocs fonctionnels qui peuvent être programmés, et reprogrammés, pour effectuer virtuellement n'importe quelle fonction logique décrite par un utilisateur dans les limites des ressources qu'il contient.

Les réseaux de portes de type dispositifs à logique programmable (Programmable Logic Device, PLD) issus de la fin des années 70 ont précédé ce que nous connaissons aujourd'hui sous le nom de FPGAs, bien que lorsque les FPGAs sont devenus disponibles au milieu des années 80 ils étaient les seuls basés sur de la mémoire statique à accès direct (Static Random-Access Memory, SRAM), hautement intégrées, et volatiles. Jusqu'à la fin des années 90, leurs utilisations étaient principalement liées aux interfaces, alors que les plus gros FPGAs étaient, et le sont toujours, utilisés comme dispositif de prototypage et de vérification pour les ASICs. De plus, les FPGAs n'auraient pas pu concourir contre les ASICs sur le prix pour la performance fournie, leur domaine applicatif était alors limité et n'était pas le centre d'un produit.

Depuis ces dix dernières années, son statut a changé. Avec l'introduction des plate-formes FPGA et des systèmes sur puce programmables, les FPGAs ont de plus en plus de fonctions à l'instar des circuits intégrés discrets. en 2011, les FPGAs ont des processeurs embarqués, des émetteurs-récepteurs multi-gigabit, des convertisseurs analogique-numérique, des blocs de calcul numériques, des contrôleurs Ethernet, une capacité mémoire substantielle et bien d'autre bloc fonctionnel comme les gestionnaires d'horloge ou les multiplieurs. Ces générations très récentes voient aussi apparaître des sous familles portées sur des applications spécifiques comme l'automobile, la communication ou le militaire. Cela signifie que l'espace d'application des FPGAs s'est agrandi et que la conception avec des FPGAs requiert désormais de la spécialisation puisqu'il ne sont plus des réseaux de simples blocs logiques.

Concernant la performance et la consommation en puissance, les FPGAs sont communément inférieurs aux ASICs, mais peuvent concurrencer en étant disponibles rapidement, reprogrammables et fabriqués avec les dernières technologies. Ils fournissent une alternative attirante lorsque les ressources (le coût, le temps) demandées par le développement ASIC ne sont pas disponibles. La possibilité de paralléliser les opérations et d'exécuter des fonctions personnalisables les rend également compétitif par rapport au microprocesseur séquentiel.

En terme de sécurité, la croissance de la capacité des FPGAs et de l'espace d'application à deux implications principales. Premièrement, les conceptions FPGA représentent un investissement qui demandent de la protection ; et deuxièmement, les FPGAs sont de plus en plus utilisés pour des applications qui demandent des propriétés de sécurité relatives aux FPGA qui sont peu ou pas disponibles aujourd'hui. Une attention toute particulière des attributs de sécurité des FPGA sera sans doute vu puisqu'ils sont désormais utilisés dans les industries militaires, automobiles ou de consommateur, chacune ayant ses propres exigences de sécurité.

Dans la communauté académique, la recherche en matière de sécurité FPGA n'a cessé d'augmenter bien que l'intersection ingénierie et sécurité est problématique. Ceci est peut être dû au fait que la sécurité n'est pas un problème d'ingénierie traditionnel et qu'elle ne possède pas des spécifications que l'on peut montrer comme étant remplies. Un système ne peut donc être prouvé sécurisé simplement parce qu'il est implémenté et qu'il marche. Ce

que nous voyons parfois publier comme schémas de sécurité peut apparaître sécurisé mais montre des failles ou est sécurisé de façon incomplète ou même impossible à implémenter en pratique.

Motivation et contribution

Cette dissertation se propose d'évaluer l'utilisation de FPGAs au sein de systèmes embarqués et de faire des propositions pour la sécurité compatible haut-débit. Le but est ici de fournir une solution matérielle efficace et transparente pour l'utilisateur. Les primitives de sécurité et autres propositions devront supporter des services de sécurité et d'aide, en adéquation avec les besoins actuels qui sont fortement liés aux réseaux. Le challenge final est simple : offrir des solutions efficaces qui correspondent formellement aux ressources restreintes des systèmes embarqués.

Le Chapitre 1 introduit les notions essentielles autour des systèmes embarqués et de la sécurité. Les menaces, faiblesses et attaques spécifiques aux cibles de ce document, forment le coeur de ce premier écrit.

Le Chapitre 2 se concentre sur le bon choix des primitives de sécurité qui sont nécessaires pour répondre aux besoins communs des systèmes embarqués. Spécifiquement, l'adaptabilité de nombreux modes de chiffrement et d'authentification sera évaluée pour le haut-débit.

Le Chapitre 3 est dérivé de la publication "Efficient Key-Dependent Message Authentication in Reconfigurable Hardware" et décrit l'implémentation optimisée d'une fonction d'authentification de message compatible haut-débit pour les systèmes embarqués.

Le Chapitre 4 est issu de la publication "Configurable Memory Security in Embedded Systems" où une approche de sécurité configurable supportant différents niveaux de sécurité est proposée. Cette gestion a pour but de maintenir des performances adéquates au système (surface, latence, mémoire).

Le Chapitre 5 provient des publications "End-to-End Bitstreams Repository Hierarchy for FPGA Partially Reconfigurable Systems" et "UDP Partial Bitstreams Diffusion Through WLAN". Une plate-forme de reconfiguration

dynamique partielle est proposée et les atouts d'une telle architecture dans le cadre de la sécurité sont discutés.

Le Chapitre 6 met en application une structure probabiliste et compacte pour stocker efficacement le matériel cryptographique. Cette nouvelle approche réduit considérablement la mémoire sur-puce et hors-puce nécessaire. Les effets indésirables de cette solution sont également mis en lumière au travers de discussions concrètes.

L'unique Annexe en fin de document, étend les propositions de cette thèse pour une approche multicoeurs. Largement expérimental, ce chapitre donne de nombreux résultats d'implémentations et montre la viabilité des contributions variées provenant de cette thèse.

Lecture de la dissertation

La lecture du document se fait de la façon la plus classique qu'il soit. Certaines pensées, informations supplémentaires ou autres interrogations sont toutefois manifestées par des boites de texte au fond gris. Elles sont pour la plupart informelles. Le document papier inclus également un CD-ROM d'accompagnement qui comprend les publications issues de ces recherches et les matériels d'implémentation en relations avec ces travaux.

Chapitre 1

Sécurité et Systèmes Embarqués

Nous proposons dans ce premier chapitre, d'établir les idées primordiales de ce que sont les systèmes embarqués autour d'une libre discussion sur les tendances passées et à venir. Comme chacun le sait, la sécurité est désormais un enjeu majeur dans notre économie, d'où le besoin de revenir sur les concepts et termes liés à la sécurité de façon à se prémunir efficacement contre des adversaires malveillants. Un premier écrit qui se veut introductif avant de procéder à une analyse plus fine de la thématique de cette thèse.

Ce chapitre est distribué en six parties. La première est une discussion autour de l'évolution des systèmes embarqués sur laquelle repose la deuxième partie dédiée aux problèmes inhérents de la sécurité. La troisième et quatrième parties proposent de mettre en lumière les challenges de la sécurité et les menaces potentielles vis-à-vis des systèmes embarqués. La cinquième est plus spécifiquement attribuée aux attaques sur les conceptions FPGA laissant découler l'habituelle conclusion en partie numérotée six.

Sommaire

1.1 Systèmes embarqués

L'engouement autour des systèmes embarqués est principalement le fruit d'un ingrédient qui n'a finalement été ajouté que récemment : le réseau. Le réseau offre en effet du confort à l'informatique embarquée, ce que le Web a apporté à Internet. Interconnectés, notamment de manière sans-fil, les systèmes embarqués deviennent alors plus faciles et plus intéressants à utiliser. Au-delà de l'aspect réseau, d'autres facteurs aussi bien technologiques que sociaux, comme les avancées concernant les processeurs basse consommation, le Web, et la demande en constante augmentation pour des services personnalisés, sont aussi responsables d'avoir stimuler les technologies embarquées et leurs intérêts pour elles.

Il est fort à parier que les systèmes embarqués ne perdrons désormais plus leur place première dans notre quotidien. Simplement parce que leur évolution est la réalisation d'un rêve longtemps attendu de rendre l'ordinateur invisible et omniprésent.

L'un des premiers challenges à l'heure actuelle est sans aucun doute la définition d'un modèle de calcul distribué pour les systèmes embarqués et connectés en réseau. Le but ultime étant de les faire coopérer pour combiner ou récolter leurs fonctionnalités ou ressources. Leurs nombres et leurs types est si grands, que les modèles distribués traditionnels ne peuvent tout simplement pas être appliqués sans poser des pénalités de programmation trop fortes. Pour conserver la programmation à une échelle raisonnable, le nouveau modèle de calcul doit tolérer résultat et synchronisation partiels ainsi qu'une faible cohérence. Les chercheurs ont proposé des solutions de mise à l'échelle pour des tâches simples et coopératives comme le routage en utilisant l'adressage basé sur le contenu.

L'interopérabilité est un second challenge. La diversité des dispositifs embarqués sur laquelle nous sommes dépendants rend notre vie plus complexe, si nous ne pouvons pas faire coopérer silencieusement ces dispositifs pour leur faire échanger des données et des tâches.

Enfin, la tolérance aux fautes et la sécurité sont des challenges traditionnels, quel que soit le système distribué, et ils déterminent l'acceptation des technologies embarquées par la société.

Les systèmes embarqués ont déjà été utilisés depuis longtemps, mais ils n'ont été connectés que sporadiquement. Premièrement, les systèmes embarqués en réseaux qui affecteront nos vies apparaîtront dans nos maisons, probablement dans nos cuisines, et seront connectés à Internet. Le premier jalon que les technologies embarquées devront poser est clairement situé dans l'industrie automobile. Tout le monde utilise des voitures et attend de voir la conduite facilité au quotidien. Il est possible d'accomplir cela sur l'ordinateur embarqué de la voiture qui coopère avec ses autres pairs et les autorités d'informations sur la route. Les applications médicales ne seront pas en reste, mais cela prendra probablement du temps avant que les systèmes embarqués se retrouvent dans les hôpitaux ou dans le corps humain, problème de bioéthique.

Dans le passé, les dispositifs embarqués étaient typiquement liés aux standards et la tendance semble vouloir suivre cet héritage. En général, les standards sont à la fois une volonté et un obstacle en informatique. Étant donné le nombre varié de systèmes embarqués et le désir d'interopérabilité, il est difficile d'imaginer une évolution sans standard. Le fait que les matières premières électroniques incorporent désormais des systèmes embarqués est un signal fort que l'industrie demande des standards.

Il est cependant peu probable que tous les standards actuels survivront aux prochaines générations de systèmes embarqués. Pourtant la stratégie marketing actuelle est fortement liée à la vente de standards traditionnels en promettant des extensions spécifiques aux technologies des systèmes embarqués. Cependant, les caractéristiques de ces standards existants sont assez différents des besoins réels des systèmes embarqués. Le protocole IP par exemple, n'est pas approprié pour certains réseaux embarqués puisque maintenir des millions de noeuds volatiles qui utilisent tous des adresses individuelles n'a pas de sens. En lieu et place d'un tel protocole, un schéma d'adressage sans protocole internet (Internet Protocol, IP) sur des propriétés ou du contenu est préférable.

L'industrie contrôle d'ors et déjà le marché du système embarqué qui à l'inverse du Web, requiert des investissements substantiels, de larges infrastructures, et un engagement à long terme. La demande est si grande que seul quelques acteurs peuvent encore jouer dans cette cour, où les alliances entre

les sociétés possédant des capitaux important et les fabricants de systèmes embarqués sont la coutume.

Les systèmes embarqués sont une plate-forme idéale pour des approches orientées composants. A l'avenir, les nouvelles générations des systèmes embarqués seront programmables et reconfigurables, ce qui incite aux conceptions simples et flexibles.

Le besoin de la sécurité émerge de bien des endroits (Ravi *et al.*, 2004b) (Ravi *et al.*, 2004a). Comme évoqué précédemment, les différentes entités peuvent communiquer entre elles et ce point particulier mène à des problèmes de sécurité. L'échange de données peut être effectivement sensible d'où un besoin de chiffrement et de preuve d'intégrité par exemple. Le temps et les fonds investis dans la conception de propriétés intellectuelles matérielles ou logicielles, ainsi que leur valeur intrinsèque, sont des facteurs qui impactent fortement sur la sécurité globale d'un système.

1.2 Sécurité

La sécurité est une nécessité pour l'industrie de l'embarqué ou l'intégrité du système, son contenu et sa connectivité gagnent l'attention du marché (Hu, 2010). Ceci mène naturellement à la prise en compte de la sécurité comme un critère qui doit être partagé dans le coût global des conceptions. Dans ce domaine, la prévalence toujours actuelle du logiciel montre qu'il est nécessaire de fonder une conception avec raison, dans le but de garantir une sécurité complète de bout-en-bout.

Le marché global dans lequel nous sommes immergés n'est pas seulement sujet aux opportunités mais également à de nouvelles menaces. Celles-ci peuvent aller de la contrefaçon à l'espionnage et sont rencontrées par de grands groupes ou des gouvernements. Ces menaces peuvent avoir des conséquences à long terme bien au-delà de l'aspect financier puisqu'il peuvent largement impacter la sécurité personnelle.

Il est vrai que le désir de constituer des profits sans investissement de départ peut sembler dans un premier temps une explication forte. Cependant, il est à noter que les gouvernements sont bien plus intéressés par l'apprentis-

sage de nouvelles connaissances leur permettant de participer à leur propre développement. L'avènement de la sophistication permet, en cela, de déployer une large palette de tentatives d'exploitation de faiblesses potentielles.

Les produits commerciaux peuvent être obtenus rapidement soit par des moyens légitimes ou par le vol. Les dispositifs militaires peuvent être obtenus par espionnage ou la capture d'équipement sur un champ de bataille. Cette facilité d'accès guide tout à chacun, à prendre au sérieux la conception de l'aspect sécurité.

L'utilisation désormais très importante des FPGAs dans les produits et systèmes de tout type demande une précaution toute particulière dans la protection de la propriété intellectuelle (IP). Son importance est typiquement du même ordre que celle des données qui sont traitées au sein du FPGA. Une source importante et pionnière concernant la sécurité pour cette technolgie peut être trouvée dans (Drimer, 2009).

Conscient de cette large problématique, la sensibilité aux menaces de sécurité a grandi. La communauté américaine a su y répondre par l'intermédiaire d'un ensemble de politiques et de standards qui sont le plus souvent des lignes de marquages fortes :

- **DoD IA** : Le département de la défense (Department of Defense, DoD) de l'assurance d'information (Information Assurance, IA), ou plus à proprement parlé, cyber, identité et de l'assurance d'information (Cyber, Identity and Information Assurance, CIIA). Il s'agit d'un ensemble de politiques, de standards et de pratiques pour protéger et défendre les informations de défense des systèmes d'informations. L'un des buts de la stratégie est de protéger les données et plate-formes de confiance et s'applique au matériel.

- **DoD/DoDD 5200** : Directive DoD 5200.1-M, Programme d'Acquisition des Systèmes de Protection, est un manuel prescrivant les standards, critère et méthodologie pour protéger contre la perte et la divulgation non autorisées des programmes d'informations essentielles, des technologies et des systèmes (Essential Program Information Technologies and Systems, EPITS). C'est une directive qui conduit le développement de capacités anti-traffic des programmes DoD.

– **FIPS** : La norme fédérale de traitement d'information standards (Federal Information Processing Standards, FIPS) est un ensemble de standard émit par l'institut nationale des standards et technologies (National Institute of Standards and Technology, NIST) pour être utilisé par toutes les agences gouvernementales et les entrepreneurs n'entrant pas dans le cadre secret-défense (FIPS-140-2), (FIPS-197), (FIPS-180-3), (FIPS-198-1).

Il existe une vaste gamme de menace sur la sécurité des conceptions qui tentent d'exploiter des faiblesses bien précises, chacune de ces menaces ayant sa propre implication. Certaines sont des menaces aux intérêts financiers d'une société, tandis que d'autres peuvent menacer la sécurité personnelle ou nationale.

La sécurité est un critère de conception important dans beaucoup de systèmes embarqués. Ces systèmes sont souvent portables et plus facilement attaquables que de traditionnelles machines de calculs type ordinateurs de bureaux ou serveurs même si la frontière est désormais très fine. Les clés d'un système sécurisé inclues des défenses contre les attaques physiques et un support léger en terme de surface et de consommation en puissance.

1.3 Les menaces sur les systèmes embarqués

1.3.1 La rétro-ingénierie

La rétro-ingénierie consiste à prendre un produit existant que des tiers peuvent sonder en regardant la disposition de la conception, les dispositifs utilisés, en téléchargeant le firmware et en analysant les interactions entre les éléments. Avec ces informations, les adversaires espèrent reconstruire la conception initiale avec comme but d'utiliser les informations pour produire leur propre produit compétitif ou d'assister leurs futurs développements de produits. Des gouvernements peuvent utiliser ces informations pour, soit développer des contre-mesures efficaces ou produire des équipements similaires.

1.3.2 Le clonage

Pour le clonage, les acteurs n'essaient pas de comprendre et de démolir totalement une conception. Leur but est de fabriquer des copies d'un produit existant, essentiellement des contrefaçons qui peuvent être vendues pour obtenir des profits plus importants que l'acteur qui a dépensé du temps et de l'argent dans le développement et le marketing. Comme les adversaires sont moins sujet à dépenser de l'argent sur la qualité des composants et dans l'assurance de qualité, les produits résultants peuvent assombrir l'image de la société et avoir un impact sur ses finances.

1.3.3 La sur-fabrication

La forme la plus simple du vol de conception est la sur-fabrication. Avec le nombre important et grandissant de sous-traitants, un fabricant d'équipement original (Original Equipment Manufacturer, OEM) a souvent à faire avec des entrepreneurs pour la fabrication de ses produits. La conséquence est que, si ceux-ci sont peu scrupuleux, ils peuvent produire des unités supplémentaires au delà des commandes prescrites par l'OEM. Bien que cette forme de contrefaçon permet d'obtenir des unités identiques aux originaux, une analyse reste difficile.

1.3.4 La falsification

Lorsque des agents extérieurs essaient d'obtenir des droits privilégiés non-autorisés sur un système électronique. La falsification peut faire partie d'un programme de rétro-ingénierie ou peut avoir un but malicieux ou criminel par nature. Par exemple, un acteur peut essayer d'extraire des données ou un firmware, ou il peut essayer de modifier celui-ci dans le but de compromettre ou de provoquer l'arrêt du système.

1.4 Les faiblesses

1.4.1 La complaisance

Probablement la vulnérabilité la plus large au sein d'une équipe de concepteurs ou d'entreprises. Les entreprises peuvent échouer à bien considérer la sécurité de leur conception par manque, soit de temps ou soit dans la croyance

que la protection légale devrait être suffisante. Le résultat est que la sécurité de conception n'est pas considérée proprement et seules des étapes minimes sont prises en considération pour protéger la propriété intellectuelle de l'entreprise. La complaisance peut être combattue par l'éducation et la connaissance que, même si la protection légale existe, la route est longue et chère.

1.4.2 Les mesures de sécurité incomplètes

Pour se prémunir, une société pourrait implémenter un schéma anti-falsification dans un système qui contient, par exemple, un FPGA. Le but est d'alerter les utilisateurs finaux de tentative de falsification. Mais si les mesures de sécurité ne s'étendent pas au chiffrement du bitstream du FPGA, alors le système est vulnérable aux menaces de type rétro-ingénierie. De plus, une attention toute particulière doit être portée, non seulement sur le dispositif ou le FPGA, mais également sur la carte et à des niveaux systèmes, en considérant des menaces potentielles à chacun de ces niveaux. Les conceptions doivent être revues et corrigées pour vérifier la bonne tenue aux critères de sécurité.

1.4.3 Les portes dérobées

Les structures implémentées dans une conception dans le but d'aider au débogue peuvent laisser des trous de sécurité. Analogue aux systèmes logiciels qui laissent des portes dérobées pour accéder aux administrations systèmes. Par exemple, si laissé dans une conception, un module de débogue peut être utilisé pour contourner la sécurité ou les dispositifs anti-falsification. Bien qu'utile pendant les phases de développement, de telles portes dérobées doivent être retirées de la conception avant la production finale.

1.4.4 Les défections de conception

Des états illégaux ou non testés peuvent exister dans une conception ce qui la rend vulnérable. La conception doit être testée, durant toute sa phase de conception, et ces états illégaux examinés avant l'obtention de la conception finale.

1.4.5 Les défections de dispositif

Similaire aux défections de conception, un dispositif peut avoir des défauts de fabrication pouvant le rendre vulnérable à une attaque. La sélection de schémas de test complet avec procédures avancées d'assurance de qualité peut minimiser ce risque.

1.4.6 Les single-event upsets

Les évènements parasites (Single Event Upset, SEU) arrivent lorsque les structures mémorisantes d'un dispositif ont leurs états altérés par l'impact de particules hautes énergies type neutrons. Un SEU peut altérer les fonctionnalités du dispositif et potentiellement compromettre la sécurité des structures présentes au sein du système. L'implémentation des techniques d'atténuation ne neutralise pas totalement cette possibilité mais augmente la fiabilité du système.

1.5 Attaque sur les conceptions à base de FPGAs

Puisque ce mémoire se propose de répondre aux problèmes de sécurité posés par les systèmes embarqués sur FPGAs, ce cas très spécifique de l'embarqué est malheureusement sujet à des attaques particulières dépendantes de la technologie employée.

1.5.1 Le décodage de bitstreams

Dés qu'un acteur récupère un bitstream, il peut essayer de le décoder pour récupérer la netlist originale. Bien que le format d'un bitstream est public, la corrélation entre les bits du bitstream et la logique ne l'est pas. Les détails de la génération du bitstream sont propriétaires et aucune information ne peut être utilisée pour retrouver la netlist à partir d'un bitstream. Étant donné la taille des FPGAs modernes et le nombre de bits de configuration impliqués, retrouver une conception complète à partir d'un bitstream est peu probable. En général, la génération du bitstream utilise des techniques d'obscurcissement.

1.5.2 L'usurpation

Un cas particulier de falsification est l'usurpation. Elle apparaît lorsqu'un agent extérieur remplace tout ou une partie du bitstream d'un FPGA ou d'un programme d'un microprocesseur avec le sien soit pour de la rétro-ingénierie soit dans le but de compromettre le système ou son infrastructure.

1.5.3 Le trojan

Pour obtenir l'accès dans un système, un acteur peut tenter d'insérer sa propre logique au sein de la conception. Le but peut être d'accéder aux données stockées dans le FPGA, d'obtenir des connaissances sur le système ou même détourner le système. Potentiellement, la logique malicieuse peut être insérée dans un système de production, ou dans la conception elle même qui peut être compromise pendant les diverses phases de développement.

1.5.4 Le readback

Le readback autorise les utilisateurs à lire les données du bitstream chargés sur le FPGA. Cette technique peut être utilisée pour vérifier la programmation du composant et à des fins de débogue. Le bitstream readback, à l'instar du bitstream de configuration, ne contient pas d'en-tête et de pied de page. Cependant, il contient des informations supplémentaires dans les éléments de mémorisation utilisateurs et potentiellement l'état courant des cellules logiques internes. Cette caractéristique est une préoccupation importante pour la perspective de sécurité puisque des données opérationnelles peuvent être retrouvées.

1.5.5 Les canaux cachés

Pour une attaque par canaux cachés, un adversaire essait d'utiliser des caractéristiques opérationnelles de la conception, par exemple la consommation en puissance pour retrouver les clés secrètes, regarder comment injecter des fautes ou avoir un aperçu de la conception.

1.5.6 L'injection de fautes

Ce type d'attaque tente de provoquer le mauvais fonctionnement d'un circuit dans le but de le forcer dans un mode de test ou de debug, dans

un état invalide, ou de faire sortir les données secrètes par l'introduction des défaillances (glitches). Les acteurs font opérer le système en dehors de ses conditions nominales en variant les entrées d'horloges, en forçant les entrées aléatoirement ou en faisant varier la tension ou la température. Avec un FPGA, ce type d'attaque peut aussi inclure la modification de bits du bitstream de configuration pour en affecter la fonctionnalité. L'injection de fautes peut aussi être réussie contre les systèmes à base de microprocesseur. Une défaillance peut provoquer le contournement d'étapes du code. Les techniques de conception modernes tentent de définir totalement tous les états possibles et effectuent des analyses de défaillances.

1.6 Conclusion

Les systèmes embarqués ont connu un essor important ces dernières années propulsés au rang d'incontournable par la mise en réseau. Si de tels systèmes sont technologiquement, commercialement et économiquement des acteurs majeurs, nous voyons apparaître deux problématiques de poids : celle de les protéger contre un nombre conséquent d'attaques, défi qui, nous le verrons, est extrêmement complexe et sujet à discussion, et celle de pouvoir les rendre compatibles avec les nouveaux et futurs standards qui tendent vers du haut voir très haut-débit. Pour se faire, une étude au mieux des services de sécurité dont il faudra se parer semble bien plus que nécessaire.

Chapitre 2

Du Bon Choix des Primitives de Sécurité

Vastes sont les problèmes soulevés dans l'étape du choix des bonnes primitives de sécurité, autant pour le concepteur du logiciel que pour celui du matériel. Ce chapitre ne sera pas s'en rappeler les cours de sécurité qu'auront pu recevoir le lecteur puisque nous reviendrons sur les grands principes de la cryptographie. Pourtant tout en développant exhaustivement les aspects nécessaires pour la bonne lecture du document, nous proposons une évaluation concrète et raisonnable des critères poussant à la bonne sélection des primitives.

Ce chapitre est divisé en six parties. Nous articulons la première section autour des différents services de sécurité puis dans les deux sections suivantes nous évoquons les blocs et les différents modes de chiffrement. La section quatre évoque les fonctions de hachage et est suivie d'une cinquième section sur les codes d'authentification. Le chapitre est conclu par une évaluation et discussion relative aux critères évoqués.

Sommaire

2.1 Les services de sécurité

Si l'on considère les attaques potentielles portées sur un système, il est difficile d'imaginer protéger celui-ci dans son ensemble et la pratique est souvent jugée comme impropre. D'une part parce que la multiplicité des failles envisageables est souvent sans scrupule envers un système, depuis son idée même, jusqu'à son emploi par l'utilisateur final et d'autre part, et l'actualité nous le montre souvent, la cryptographie qui sait pourtant répondre à ces problèmes est souvent mal, voire très mal utilisée. Tôt ou tard surviendra l'irrémédiable faille à une échelle plus ou moins importante, que l'on pourra corriger ou non. L'escalade entre les attaques et les parades est certaine si une étude sérieuse n'est pas menée. La question primordiale qui doit être posée est la suivante : quels sont les services de sécurité nécessaires pour un système donné ? Pour la plupart des applications, l'on considère 4 principaux services désirables :

1. **La confidentialité** : l'information est gardée secrète de tous les tiers non autorisés.

2. **L'intégrité** : l'information n'a pas subi de modification.

3. **L'authentification** : l'émetteur de l'information est authentique.

4. **La non répudiation** : l'émetteur de l'information ne peut dénier sa part dans un échange.

Dans ce chapitre, nous proposons de revenir sur les aspects étendus de la cryptographie et notamment sur le choix des différentes primitives pour assurer de tels services. Le but visé est de produire une référence pour une évaluation claire et concise, mais surtout multi-critères adaptée aux cibles potentielles, qui sont, dans ce document, liées à l'embarqué. L'ouvrage (Paar et Pelzl, 2010) est un référence forte dont est en partie issu ce chapitre.

2.2 Blocs de chiffrement

Le bloc de chiffrement fait parti de l'une des briques de bases en cryptographie. Si il est bien évidemment coutume de l'employer pour son rôle premier, i.e. le chiffrement, il n'est pas peu commun de le voir utiliser de manière plus exotique. Ainsi, les blocs de chiffrement sont intéressants pour réaliser des chiffrements de flux, des fonctions de hachage, pour construire des codes d'authentification de message (Message Authentication Code, MAC), pour établir des protocoles de diffusion de clés, ou encore pour réaliser des générateurs de nombres pseudo-aléatoires. C'est donc sur cette première sélection que s'orientera la fiabilité et la robustesse de la sécurité finale d'un système. Il est donc essentiel de valider un choix en amont de toute conception.

2.2.1 DES et 3DES

C'est le candidat Lucifer, issu d'une équipe de cryptographes d'IBM, qui est retenu pour être le standard de chiffrement symétrique de données (Data Encryption Standard, DES (FIPS-46-3)) en 1977. Il s'agit d'une famille de chiffrement développée par Horst Feistel (inventeur des réseaux qui portent son nom) vers la fin des années 60 et qui fut la première à opérer sur des données numériques. A l'origine Lucifer chiffre des blocs de 64 bits et utilise une taille de clé de 128 bits. Il est à souligner que sa version DES a subi des transformations concernant cette taille, passant de 128 à 56 bits. Les raisons de ces changements sont assez floues, d'autant plus que la robustesse du bloc de chiffrement est directement liée à cette grandeur.

A l'origine, avec l'aide de l'agence nationale de sécurité (National Security Agency, NSA), le bureau national des standards (National Bureau of Standards, NBS) maintenant nommé NIST, proposa un appel à contribution pour définir un des tout premier standard de chiffrement. Cet appel fut poussé principalement par la demande progressive de chiffrement dans les applications commerciales et le secteur bancaire.

Les deux attaques analytiques principales contre DES, la cryptanalyse différentielle et linéaire, font aujourd'hui parties des deux méthodes généralistes les plus puissantes pour disqualifier un bloc de chiffrement (Biham et Shamir, 1993) (Matsui, 1994). A cause de son faible espace de clés, DES devrait donc être inutilisé, puisque la puissance de calcul désormais disponible,

permet une attaque par force brute pour un faible coût et ce, en un temps marginal. Citons les deux machines COPACOBANA (Kumar *et al.*, 2006) (Güneysu *et al.*, 2008) et Deep Crack (Foundation, 1998), comme exemples réputés et dédiés à cette tâche.

Une version augmentée de DES par triplement, 3DES, permet d'obtenir un bloc de chiffrement pour lequel aucune attaque pratique n'est connue. Cette version utilise certes 3 clés de 56 bits, mais la robustesse en sécurité de l'algorithme n'est pas cumulée. Ainsi par l'attaque de la rencontre au milieu, la force effective est de 112 bits et non pas de 168 bits.

Le bloc de chiffrement DES fut le bloc de chiffrement le plus populaire de ces 30 dernières années. Aucune faiblesse ne fut trouvée avant 1990 et son arrêt fut finalement programmé par l'adoption du standard AES.

2.2.2 AES

Le bloc de chiffrement standard symétrique et avancé (Advanced Encryption Standard, AES) est un standard mondial et l'une des primitives cryptographiques des plus populaires. Il est largement employé dans les standards industriels et utilisé dans bien des applications commerciales. AES est également une pièce maîtresse des standards de sécurité Internet comme le protocole de sécurité Internet (Internet Protocol security, IPsec), le protocole sécurisé d'échange (Transport Layer Security, TLS), l'accès sans-fil et protégé (Wi-Fi Protected Access, WPA) ou bien encore le logiciel et protocole de communication sécurisé (Secure SHell, SSH). Cette large adoption peut être expliquée par des implémentations efficaces à la fois en logiciel et en matériel.

C'est en 1999 que la NIST propose le remplacement de DES au profit de 3DES. Même si la résistance de 3DES est toujours d'actualité contre les attaques par force brute, sa faible efficacité pour une implémentation logicielle ainsi que la taille du bloc (64 bits), en on fait un algorithme déprécié. La NIST appelle alors publiquement à soumission en 1997 pour définir le futur standard de chiffrement AES. En 2001, c'est le candidat finaliste Rijndael (défini dans le document (FIPS-197)) conçu par Joan Daemen et Vincent Rijmen, qui remporte la mise. Contrairement à l'algorithme DES, AES utilise des tailles de clés de 128, 192, et 256 bits, et n'est pas basé sur les réseaux de Feistel mais sur l'arithmétique dans les corps de Galois.

Conçu à l'origine en 1997, AES a survécu aux nombreux efforts de cryptanalyse et aucune attaque autre et meilleure que l'approche force brute n'est connue. Bien que de nombreux papiers ont été publiés sur la cryptanalyse de l'AES, l'attaque simple clé la plus rapide sur des variantes AES avec rondes réduites (Dunkelman *et al.*, 2010) (Mala *et al.*, 2010) ne montre qu'une efficacité légèrement supérieure à celles proposées 10 ans plus tôt (Ferguson *et al.*, 2001) (Gilbert et Minier, 2000). Quelle que soit la version AES le nombre de rondes cryptanalysées n'a pas augmenté depuis (7 pour une clé de 128 bits, AES-128, 8 pour AES-192 et AES-256). Seule une réduction de complexité de calcul requis pour l'obtention de la clé a été atteinte. En général, les dernières années ont permis de voir émerger des progrès sur la cryptanalyse des blocs de chiffrement. Cependant, le bloc de chiffrement standard AES est presque aussi sûr qu'il était il y a 10 ans dans son modèle le plus fort et le plus pratique avec une simple clé non connue. L'ancien standard DES n'a pas fait l'objet d'amélioration majeure depuis le papier de Matsui en 1994 (Matsui, 1994).

A contrario, la discipline qu'est la cryptanalyse des fonctions de hachage a rapidement grandi, encouragée par la cryptanalyse de résumé de message (Message Digest 5, MD5) (Wang et Yu, 2005), de l'algorithme de hachage sécurisé (Secure Hash Algorithm, SHA-0) (Biham *et al.*, 2005) (Chabaud et Joux, 1998), et SHA-1 (Wang *et al.*, 2005) suivi d'attaques pratiques sur les protocoles utilisant MD5 (Stevens *et al.*, 2007) (Stevens *et al.*, 2009), des attaques pré-image sur Tiger (guo, 2010), et MD5 (Sasaki et Aoki, 2009), etc. Comme la cryptanalyse différentielle (Biham et Shamir, 1991), technique développée à l'origine pour les blocs de chiffrements, à été portée pour l'analyse de fonction de hachage, les cryptanalystes cherchent maintenant l'opposé : une méthode issue de l'analyse des fonctions de hachage qui donnerait des nouveaux résultats pour les blocs de chiffrement. La tentative la plus connue est l'analyse d'AES avec des collisions locales (Biryukov *et al.*) (Biryukov et Khovratovich, 2009) (Biryukov *et al.*, 2009) (Biryukov et Nikolić, 2010). Cependant, de nombreuses hypothèses fortes sont prononcées et sont rarement vérifiées en pratique. Elles ne sont donc pas considérées comme étant une menace dans l'utilisation d'AES.

AES a été conçu pour faire face aux cryptanalyses différentielles et linéaires (Daemen et Rijmen, 2002). De telles techniques (dans leurs versions

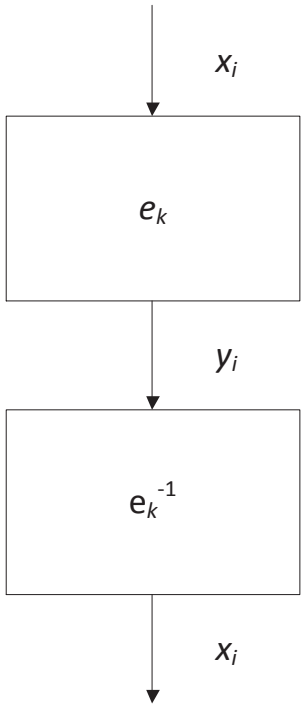

FIGURE 2.1 – Chiffrement et déchiffrement du mode ECB

pures) sont limitées dans le cadre d'applications aux attaques. Les méthodes d'obtention de clés les plus puissantes sont la cryptanalyse différentielle dite impossible (Biham *et al.*, 1999) (Mala *et al.*, 2010) et les attaques carrées (Daemen *et al.*, 1997) (Dunkelman *et al.*, 2010). La cryptanalyse différentielle impossible a donné lieu à la première attaque sur un AES-128 de 7 rondes. L'attaque carrée et ses variations comme l'attaque intégrale et l'attaque multi-ensemble fournissent une cryptanalyse de variantes AES avec des complexités en calcul les plus faibles à l'heure actuelle, tandis que la première attaque sur une version 8 rondes d'AES-192 (Dunkelman *et al.*, 2010) n'est apparue que très récemment.

Les attaques de l'homme du milieu sur les blocs de chiffrement ont perçu moins d'attention (quelques exceptions cependant (Dunkelman et Keller, 2010) (Dunkelman *et al.*, 2007) (Bogdanov et Rechberger, 2011) (Isobe, 2011) (Wei *et al.*, 2011)) que les approches différentielles, linéaires, différentielles impossibles, et intégrales. Cependant, ce sont probablement les plus praticables en terme de complexité de données. Une simple attaque de l'homme du milieu ne requiert qu'une seule paire texte clair/texte chiffré. L'utilisation limitée de ces attaques est attribuée à la nécessité de l'indépendance du bloc de chiffrement vis-à-vis des bits de la clé. Pour un bloc de chiffrement avec un ordonnancement non linéaire, comme AES et la plupart des candidats AES, il s'agit là d'une forte considération. Il en résulte un nombre réduit de rondes compromises (Dunkelman et Keller, 2010), et ceci semble montrer que sur un nombre de rondes supérieures (8, 9 et 10), la technique n'est pas appropriée.

AES est donc un algorithme robuste et éprouvé d'un point de vue de la

Utilisation des IVs : il est important de relever que la façon de procéder à la génération des vecteurs d'initialisation (IV) n'est pas unique mais que des règles bien précises se doivent d'être respectées pour garantir une sécurité optimale. Peter Gutmann, chercheur à l'université de Auckland, donne cependant un éclaircissement sur la nécessité de sensibiliser sur ce point : "Je suis préoccupé par le fait que les personnes qui lisent les publications en cryptographie (i.e les personnes en sécurité) n'en n'auront probablement jamais besoin, et que ceux qui en ont vraiment besoin (les développeurs), ne les lisent jamais, voire même pire, il ne savent même pas qu'elles existent.".

sécurité. Il possède des propriétés intéressantes lui permettant des implémentations optimisées en logiciel et matériel.

2.3 Les modes de chiffrement

A partir d'un bloc de chiffrement, AES ou DES par exemple, différents fonctionnements de chiffrement sont offerts. Ces multiples façons de procéder, possédant avantages et inconvénients, sont nommées mode de chiffrement ou mode d'opération.

2.3.1 Mode ECB

Le mode dictionnaire de codes (Electronic Code Book, ECB) est le plus simple et le plus direct pour chiffrer un message et est appelé, par son comportement, chiffrement déterministe. Considérons $e_k(x_i)$ comme étant le chiffrement de texte clair x_i avec une clé k et $e_k^{-1}(y_i)$ le déchiffrement du texte chiffré avec cette même clé. Considérons également que le bloc de chiffrement chiffre des blocs de taille b bits. Le processus de chiffrement est montré en Figure 2.1 et est décrit formellement comme suit :

$$Chiffrement \quad : \quad y_i = e_k(x_i), i \geq 1 \tag{2.1}$$
$$Déchiffrement \quad : \quad e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)), i \geq 1 \tag{2.2}$$
$$e_k^{-1}(y_i) = e_k^{-1}(e_k(x_i)) = x_i \tag{2.3}$$

Le mode ECB a bien des avantages. Premièrement, la synchronisation des blocs entre deux parties n'est pas nécessaire. Deuxièmement, des erreurs de bits causés par des transmissions bruitées, n'affectent que le bloc correspondant et non les blocs suivants. Enfin le mode ECB peut être parallélisé ce qui est souvent important pour obtenir des implémentations haut-débit.

Cependant, des faiblesses sont associées à ce mode et malheureusement ce cas se répète souvent en cryptographie. La principale est liée au chiffrement qui est très déterministe. De chaque bloc de texte clair résultera un bloc similaire chiffré identique, si la clé ne change pas. Un attaquant peut reconnaître aisément si un message a été transmis deux fois simplement en scrutant le texte chiffré. De plus, il peut également réordonner deux blocs

puisque chacun d'eux est chiffré indépendamment et ce, sans que cela ne soit détecté.

Il faut donc lui préférer un mode permettant d'obtenir un texte chiffré différent à chaque fois que l'on chiffre un nouveau texte clair, comportement que l'on appelle chiffrement probabiliste. C'est par notamment l'utilisation de vecteur d'initialisation (Initialization Vector, IV) que l'on introduit une source aléatoire dans ce but.

2.3.2 Mode CBC

Dans ce mode par enchaînement de blocs (Cipher Block Chaining, CBC), tous les blocs de chiffrement sont chaînés tel que le texte chiffré y_i dépend non seulement du bloc x_i mais aussi des blocs de texte clair. Le chiffrement subit également des transformations aléatoires par l'utilisation d'un IV.

Le texte chiffré y_i, résultat du chiffrement du bloc de texte clair x_i est rebouclé à l'entrée du bloc de chiffrement où est effectué un ou-exclusif avec le bloc de texte clair suivant x_{i+1}. Cette somme est alors chiffrée donnant naissance au texte chiffré suivant y_{i+1}. Ce processus est montré en Figure 2.2. En ce qui concerne le premier bloc de texte clair, il n'y a pas de texte chiffré précédent. Un IV est ajouté au premier texte clair ce qui rend le chiffrement CBC non-déterministe. Le premier texte chiffré y_1 dépend du texte clair x_1 et de l'IV. Le second texte chiffré y_2 dépend de x_1, x_2 et de l'IV, etc... . Le dernier texte chiffré est une fonction de tous les blocs de texte clair et le l'IV.

Lors du déchiffrement d'un bloc de texte chiffré y_i en mode CBC, l'opération est l'inverse du chiffrement comme montré en Figure 2.2. Les processus de chiffrement et déchiffrement sont décrits comme cela :

$$\begin{aligned}
Chiffrement &: y_i = e_k(x_i \oplus IV), i = 1 && (2.4)\\
Chiffrement &: y_i = e_k(x_i \oplus y_{i-1}), i \geq 2 && (2.5)\\
Déchiffrement &: x_i = e_k^-1(y_i) \oplus IV, i = 1 && (2.6)\\
Déchiffrement &: x_i = e_k^-1(y_i) \oplus y_{i-1}, i \geq 2 && (2.7)\\
&\quad d(y_1) = e_k^{-1}(y_1) \oplus IV && (2.8)
\end{aligned}$$

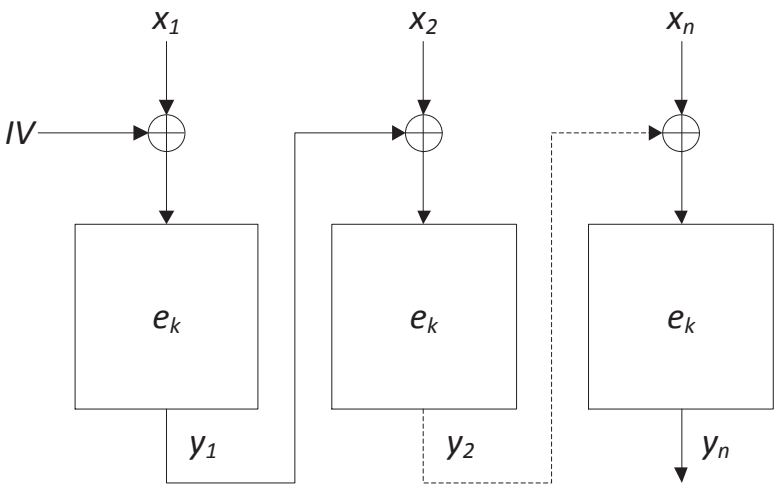

FIGURE 2.2 – Chiffrement du mode CBC

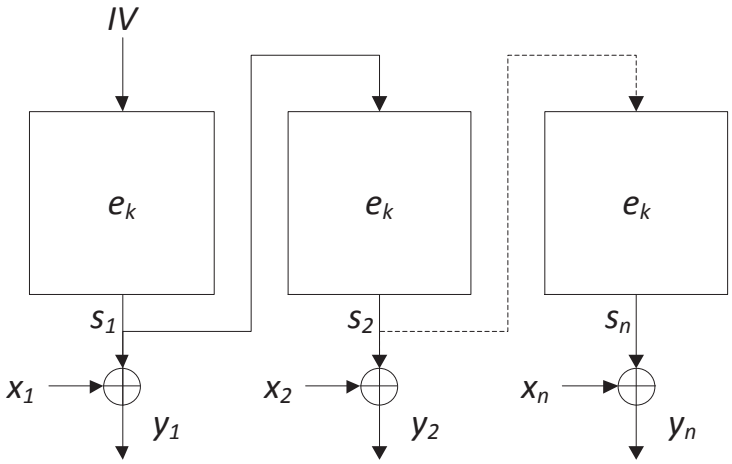

FIGURE 2.3 – Chiffrement du mode OFB

$$
\begin{aligned}
d(y_1) &= e_k^{-1}(e_k(x_1 \oplus IV)) \oplus IV \\
d(y_1) &= (x_1 \oplus IV) \oplus IV = x_1 \\
d(y_1) &= e_k^{-1}(y_1) \oplus y_{i-1} \\
d(y_1) &= e_k^{-1}(e_k(x_i \oplus y_{i-1})) \oplus y_{i-1} \\
d(y_1) &= (x_i \oplus y_{i-1}) \oplus y_{i-1} = x_i
\end{aligned} \tag{2.9}
$$

Le mode CBC est donc un mode de chiffrement probabiliste si à chaque chiffrement un nouvel IV est utilisé. Ces vecteurs ne doivent pas nécessairement être secrets. Il existe différentes manières de générer ces valeurs d'initialisation qui, dans la majorité des cas seront des nonces. L'utilisation de nombres choisis aléatoirement, un simple compteur monotone où des fonctions type LFSR sont, par exemple, des méthodes classiquement employées.

Il est intéressant de voir qu'ici, les attaques typiquement employées pour le mode ECB ne sont pas valides. Si un IV est proprement choisi pour chaque transfert, il sera impossible à un adversaire de reconnaître des motifs dans le texte chiffré.

2.3.3 Mode OFB

Pour le mode de chiffrement à rétroaction de sortie (Output Feedback Mode, OFB), un bloc de chiffrement est utilisé en tant que chiffrement de flux. En sortie du chiffrement se retrouve un flux de clé b de la taille du bloc de chiffrement sous-jacent et qui peut être utilisé pour chiffrer un texte clair de b bits avec une simple opération ou-exclusif.

L'IV est premièrement chiffré avec le bloc de chiffrement et le flux de clé en sortie de ce bloc, qui est rebouclé, se retrouve en entrée pour calculer le prochain flux de clé. Ce processus est répété comme montré en Figure 2.3. Le processus de déchiffrement, est identique à celui du déchiffrement et décrit selon :

$$Chiffrement \;:\; s_i = e_k(IV); y_i = s_i \oplus x_i, i = 1 \tag{2.10}$$
$$Chiffrement \;:\; s_i = e_k(s_{i-1}); y_i = s_i \oplus x_i, i \geq 2 \tag{2.11}$$
$$Déchiffrement \;:\; s_i = e_k(IV); x_i = s_i \oplus y_i, i = 1 \tag{2.12}$$
$$Déchiffrement \;:\; s_i = e_k(s_{i-1}); x_i = s_i \oplus y_i, i \geq 2 \tag{2.13}$$

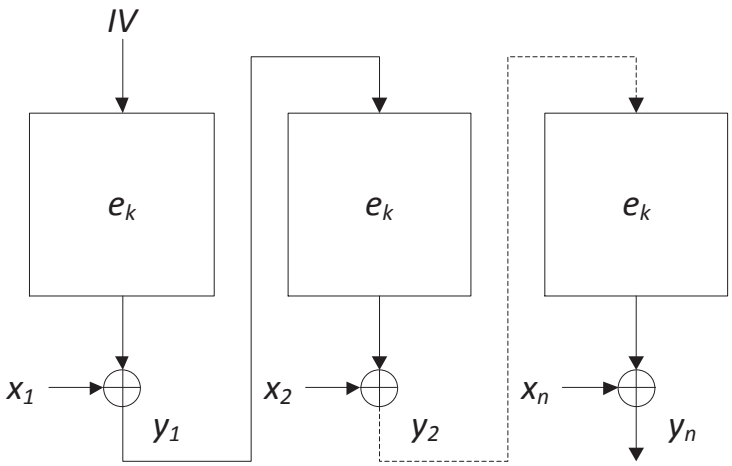

FIGURE 2.4 – Chiffrement du mode CFB

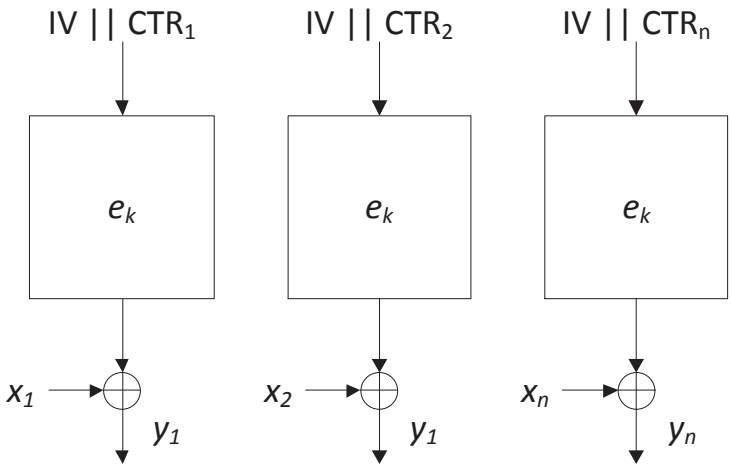

FIGURE 2.5 – Chiffrement du mode CTR

Le mode de chiffrement OFB, puisqu'il utilise un IV, est donc probabiliste. L'un des avantages de celui-ci est que les calculs du bloc du chiffrement sont indépendants du texte clair. Il est donc possible de pré-calculer plusieurs blocs de flux de clé s_i.

2.3.4 Mode CFB

Le mode de chiffrement à rétroaction (Cipher Feedback, CFB) se fonde sur la même utilisation d'un bloc de chiffrement en chiffrement de flux. Il est similaire au mode OFB. Sa seule différence se situe au niveau du rebouclage. Souvenons-nous que pour le mode OFB, c'est le flux de clé qui est rebouclé en entrée du bloc de chiffrement. Ici, c'est bel et bien le texte chiffré (donc le résultat d'un ou-exclusif entre le texte clair et le flux de clé) qui est rebouclé. La Figure 2.4, décrit plus précisément ce processus et la description formelle du mode CFB est la suivante :

$$Chiffrement \quad : \quad y_i = e_k(IV) \oplus x_i, i = 1 \tag{2.14}$$
$$Chiffrement \quad : \quad y_i = e_k(y_{i-1}) \oplus x_i, i \geq 2 \tag{2.15}$$
$$Déchiffrement \quad : \quad x_i = e_k(IV) \oplus y_i, i = 1 \tag{2.16}$$
$$Déchiffrement \quad : \quad x_i = e_k(y_{i-1}) \oplus y_i, i \geq 2 \tag{2.17}$$

Tout comme les modes CBC et OFB, le mode CFB est non déterministe.

2.3.5 Mode CTR

Le mode par compteur (CounTeR, CTR) est également basé sur l'utilisation d'un bloc de chiffrement comme chiffrement de flux. L'entrée de ce bloc est une valeur de compteur qui est supposée être différente à chaque fois que la génération d'un nouveau bloc de flux de clé a lieu. La Figure 2.5 montre ce principe. Pour assurer cette unicité, on choisi tout d'abord un IV qui est un nonce avec un taille inférieure à la taille du bloc de chiffrement (i.e. 96 bits pour un bloc de chiffrement 128 bits). Les 32 bits restants sont utilisés comme compteur initialisé à zéro. Pour chaque bloc qui sera chiffré, le compteur sera incrémenté mais l'IV reste le même. Dans ce cas précis, le nombre de blocs pouvant être chiffré avec un IV similaire est 2^{32}, i.e. $8 \times 2^{32} = 2^{35}$ octets soit environ 32 Gigaoctets. Plus formellement, voici la description des processus de chiffrement et déchiffrement du mode CTR :

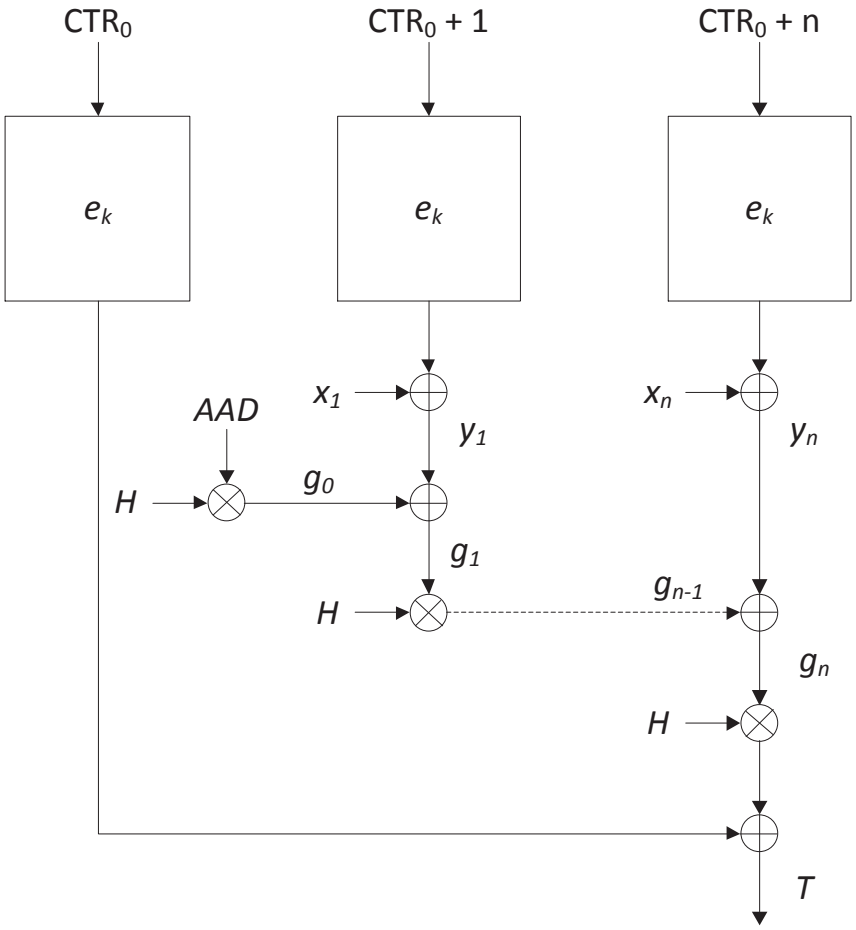

FIGURE 2.6 – Chiffrement authentifié du mode GCM

$$Chiffrement \quad : \quad y_i = e_k(IV||CTR_i) \oplus x_i, i \geq 1 \tag{2.18}$$
$$Déchiffrement \quad : \quad y_i = e_k(IV||CTR_i) \oplus y_i, i \geq 1 \tag{2.19}$$

Tout comme l'IV, le compteur CTR ne doit pas être nécessairement secret. Celui-ci est également générer de manière similaire à l'IV. Le mode compteur ne possédant pas de rebouclage, il est attrayant pour la parallélisation, caractéristique souvent nécessaire pour obtenir de haut débits. n blocs de chiffrement peuvent, par exemple, chiffrer m valeurs de compteur CTR_1, CTR_2,... CTR_m en même temps. Rajouter n blocs de chiffrement revient à augmenter le débit proportionnellement.

2.3.6 Mode CCM

Le mode compteur avec CBC-MAC CCM (Nat, 2004) est un mode qui combine le mode CTR et l'algorithme d'authentification CBC-MAC. Ce type de construction est appelé mode de chiffrement-authentifié puisqu'il fournit à la fois le service de la confidentialité et le service d'authentification. Une seule clé secrète est utilisée pour effectuer ces deux opérations et la même politique concernant l'utilisation des IVs que les modes précédemment décrits s'applique également. CCM demande deux appels au bloc de chiffrement pour effectuer le chiffrement et l'authentification d'un bloc de texte clair.

2.3.7 Mode GCM

Similairement au mode CCM, le mode de chiffrement de Galois par compteur (Galois Counter Mode, GCM) (Nat, 2007) est un mode de chiffrement-authentifié. La confidentialité est assurée par le chiffrement du texte clair en mode CTR et l'authentification est effectuée par le calcul d'un MAC fondé sur le hachage universel. Une chaîne de donnée supplémentaire additionnelle (Additional Authenticated Data, AAD) peut également être authentifiée, mais elle ne sera toutefois pas chiffrée. La pratique veut que cet élément soit utilisé pour authentifier des paramètres de protocoles réseaux comme les adresses IPs ou les ports.

GCM est constitué d'un bloc de chiffrement et d'un élément de multiplication dans les corps de Galois. Bien qu'il soit possible d'utiliser des blocs

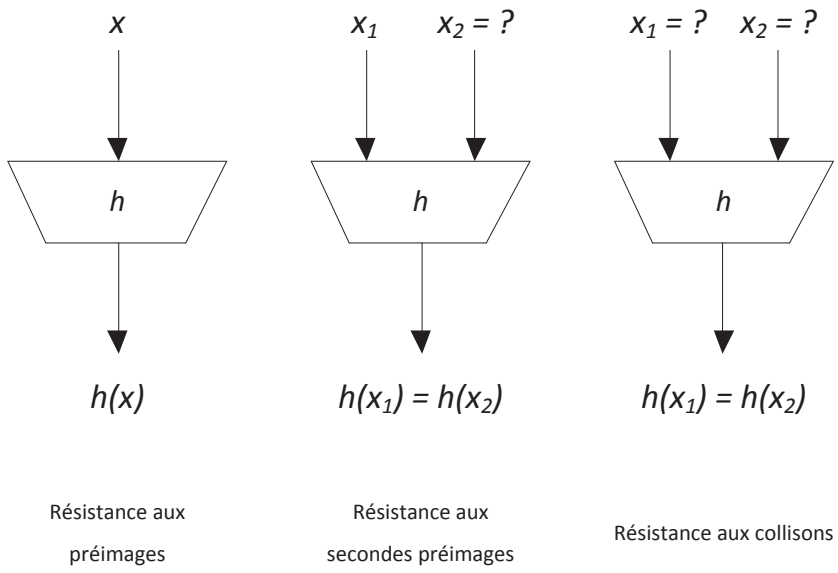

FIGURE 2.7 – Les trois propriétés de sécurité d'une fonction de hachage

de chiffrement réduit sur 64 bits il est recommandé, dans un souci de sécurité, d'employer des blocs de chiffrement sur 128 bits. Le fonctionnement du chiffrement étant similaire à celui du mode compteur, nous nous référons directement à la section précédente.

Le processus d'authentification est réalisé par des multiplications chaînées dans les corps de Galois. Pour chaque texte clair x_i, une sortie g_i est calculée, et est la somme ou-exclusif du texte chiffré y_i et de y_{i-1} multipliée par une constante H. La valeur H est une clé de hachage générée par le chiffrement d'une entrée "tout-à-zéro" avec le bloc de chiffrement. Toutes les multiplications ont lieu dans le corps de Galois 128 bits binaire $GF(2^{128})$ avec le polynôme irréductible $P(x) = x^{128} + x^7 + x^2 + x + 1$. La Figure 2.6 montre le fonctionnement du mode GCM et le chiffrement et déchiffrement est décrit comme suit :

$$
\begin{aligned}
Chiffrement &: CTR_1 = CTR_0 + 1 && (2.20)\\
&: y_i = e_k(CTR_i) \oplus x_i, \geq 1 && (2.21)\\
Authentification &: H = e_k(0) && (2.22)\\
&: g_0 = AAD \times H && (2.23)\\
&: g_i = (g_{i-1} \oplus y_i) \times H, 1 \leq i \leq n && (2.24)\\
&: T = (g_n \times H) \oplus e_k(CTR_0) && (2.25)
\end{aligned}
$$

2.4 Fonctions de hachage cryptographique

Autres primitives importantes de la cryptographie : les fonctions de hachage. Elles calculent un condensat ou haché qui est un résumé cryptographique d'un message. Leurs applications est vastes mais elles ont une part très importante dans les schémas de signatures numériques et les codes d'authentification de message. Soulignons qu'à l'inverse de tous les autres algorithmes cryptographiques, les fonctions de hachage ne possèdent pas de clé.

Montrée en Figure 2.7, une fonction de hachage doit posséder 3 propriétés pour être considérée comme sûre :

1. résistance aux préimages. La fonction doit être à sens unique.

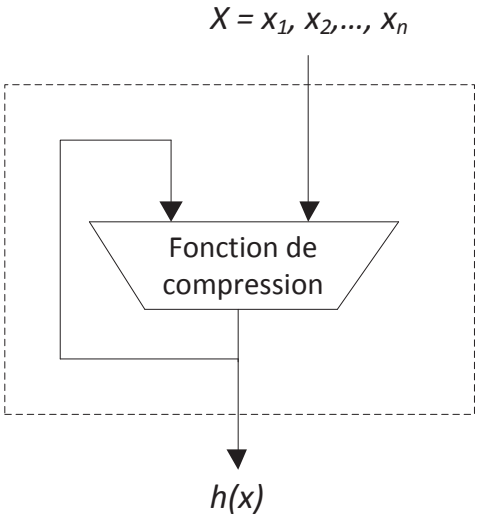

FIGURE 2.8 – Construction Merkle-Damgård

2. resistance aux secondes préimages. La fonction doit être de résistante faible aux collisions.

3. résistance aux collisions. La fonction doit être de résistante forte aux collisions.

2.4.1 Construction

Les fonctions de hachage calculent le haché d'un message de taille arbitraire et produisent des sorties de taille fixe. En pratique ceci est obtenu en segmentant les entrées en séries de blocs de même taille. Ces blocs sont traités de façon séquentielle par la fonction de hachage qui possède en interne une fonction de compression comme montrée en Figure 2.8. Une telle construction est appelée construction de Merkle-Damgård. En pratique, les fonctions de hachage peuvent être construites par des fonctions de hachage dédiées ou des blocs de chiffrement chaînés.

2.4.1.1 Fonctions de hachage dédiées

Les fonctions de hachage dédiées sont des algorithmes qui ont été spécifiquement conçus pour cette tâche. Un grand nombre de constructions à été proposé durant les 20 dernières années. De part son implémentation simple, c'est la famille de fonction MD4 spécifiée par Ronald Rivest qui porte le fondement des algorithmes comme MD5, SHA ou bien encore l'algorithme de résumé de message par évaluation de primitives pour l'intégrité (RACE Integrity Primitives Evaluation Message Digest, RIPEMD). Voyant pointer des faiblesses potentielles sur la version améliorée de MD4, MD5, la NIST publia en 1993 le nouveau standard sous l'appellation SHA référencé SHA-0.

Une seconde version, SHA-1, a rapidement été proposée pour améliorer la sécurité cryptographique. En 1996, une attaque partielle par Hans Dobbertin contre MD5 (encore largement utilisé dans la sécurité des protocoles Internet) pousse les experts à désormais recommender SHA-1. Tout comme SHA-0, SHA-1 à une taille de condensat de 160 bits et une resistance aux collisions d'environ 2^{80}. C'est par la suite que la NIST introduisit 3 variantes

Algorithme	Taille sortie (bits)	Taille entrée (bits)	Nb. de rondes	Collisions
MD5	128	512	64	oui
SHA-1	160	512	80	pas encore
SHA-224	224	512	64	non
SHA-256	256	512	64	non
SHA-384	384	1024	80	non
SHA-512	512	1024	80	non

Tableau 2.1 – Fonctions de hachage issues de la famille MD4

supplémentaires SHA-256, SHA-384 et SHA-512 qui ont des tailles de condensat de 224, 256, 384, et 512 bits respectivement. En 2005, une attaque par collision sur MD5 et SHA-0 par Xiaoyun Wang en 2004, est étendue à SHA-1 avec une recherche de collision en 2^{63} étapes bien moins que la sécurité initiale de 2^{80} par l'attaque du paradoxe des anniversaires. Le Tableau 2.1 donne un aperçu des paramètres principaux de la famille MD4.

La NIST à ouvert une compétition publique le 2 Novembre 2007 dans le but de développer une nouvelle fonction de hachage cryptographique, SHA-3, pour améliorer les algorithmes actuels spécifiés dans le document FIPS 180-3, Secure Hash Function (FIPS-180-3). Cette compétition est une réponse aux avancées en matière de cryptanalyse sur ces fonctions de hachage. 5 candidats sont retenus pour la finale qui aura lieu courant 2012.

- **BLAKE** : BLAKE (Aumasson *et al.*, 2008) suit un mode d'itération basé sur le modèle de hachage itérative (HAsh Iterative FrAmework, HAIFA (Biham et Dunkelman, 2006)) et opère sur un état intérieur pouvant être représenté par une matrice carrée 4 par 4 de mots. Cet état est initialisé par un IV, un salt et un compteur. Il est mis à jour par la fonction G qui est basée sur le chiffrement de flux Chacha (Bernstein). La fonction G met à jour les colonnes et les diagonales disjointes de l'état avec des additions modulaires, des ou-exclusifs et des opérations de rotations. Le bloc message d'entrée et les constantes sont sélectionnées par des permutations fixes dépendantes des rondes. La non-linéarité de la conception est obtenue par les additions modulaires. BLAKE est retenu comme l'un des 5 finalistes de la compétition dû à sa haute marge en sécurité, de bonne performance en logiciel et une conception simple et claire.

- **Grøstl** : Grøstl (Gauravaram *et al.*, 2008) est basé sur l'algorithme de hachage Merkle-Damgård. La fonction de compression est nouvelle et utilise deux permutations fixes de 2n-bits pour produire une fonction de compression de $2n$ bits dans le but d'obtenir une résistance aux collisions et aux attaques préimage d'une fonction idéale de compression sur n bits. L'étape de transformation de la sortie calcule et traite l'état chaîné final et défausse la moitié des bits du résultat menant à un haché en sortie de n bits. Grøstl est sélectionné comme finaliste. Sa conception est en effet compréhensible et possède de solide performance en

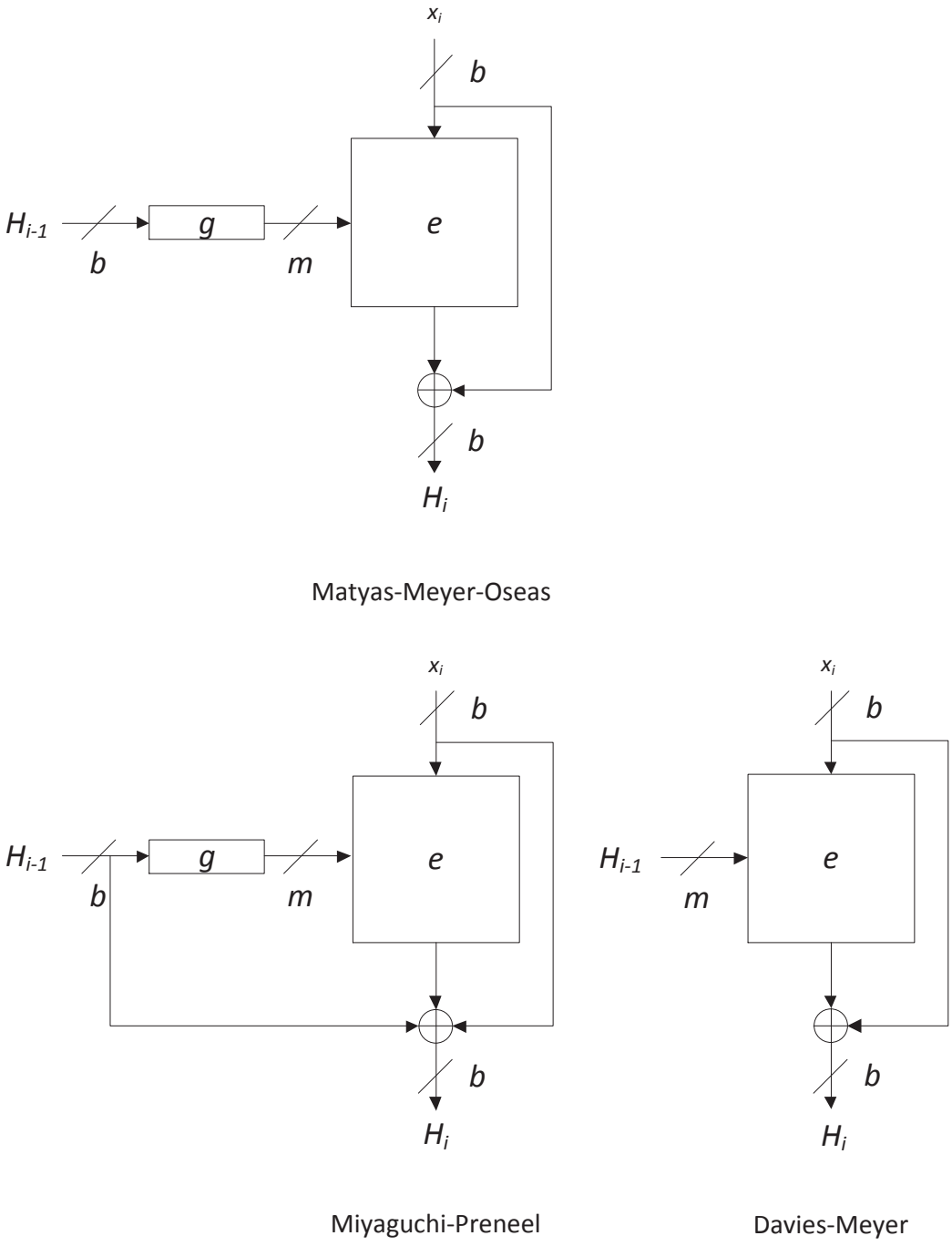

FIGURE 2.9 – Les trois constructions d'une fonction de hachage avec des blocs de chiffrement

matériel notamment. La marge en sécurité n'est cependant pas idéale. La NIST pense que cette marge pourrait être considérablement réduite par la large gamme d'attaque offerte par la cryptanalyse.

– **JH** : La famille de fonctions de hachage JH (Wu, 2009) pour différentes tailles de sorties est basée sur une unique fonction de compression F_8, qui utilise une permutation fixe E_8. Les différents membres de la famille JH sont distingués par l'utilisation de différents IV. Les valeurs de hachage peuvent être obtenues par troncation de la sortie finale. JH peut être efficacement implémenté dans un mode bitslice. JH est sélectionné comme finaliste. Par une conception innovante, JH possède une solide marge de sécurité et les performances générales sont également jugées bonnes.

– **Keccak** : Keccak (Bertoni *et al.*, 2008) suit un modèle de construction en éponge (Bertoni *et al.*, 2007). La permutation peut être considérée comme un réseau de substitution-permutation avec des S-boxes de 5-bits de large, ou comme une combinaison d'opération de mixage linéaire et une opération très simple de mixage non-linéaire. La construction de la permutation est la partie la plus innovante de Keccak. Le paramètre de sécurité recommandé pour Keccak est 34 rondes. Keccak est un finaliste par sa marge de sécurité importante, son fort débit, un bon rapport débit/surface et sa simplicité de construction.

– **Skein** : Skein (Ferguson *et al.*, 2009) est un algorithme de hachage itératif construit sur le bloc de chiffrement Threefish. Threefish est utilisé pour construire une fonction de compression pour Skein avec une version modifiée de la construction Mateas-Meyer-Oseas qui est alors itérée dans un mode chaîné similaire à HAIFA. Le mode par simple itération de bloc (Unique Block Iteration, UBI) fournit une indifférentiabilité pour un oracle aléatoire dans un modèle de chiffrement idéal. Threefish est un réseau de substitution/permutation de 72 rondes avec des fonctions de mixage de 128 bits qui consistent d'additions 64 bits, de rotations et de ou-exclusifs. Skein à été sélectionné comme finaliste par sa forte marge de sécurité et sa rapidité en logiciel.

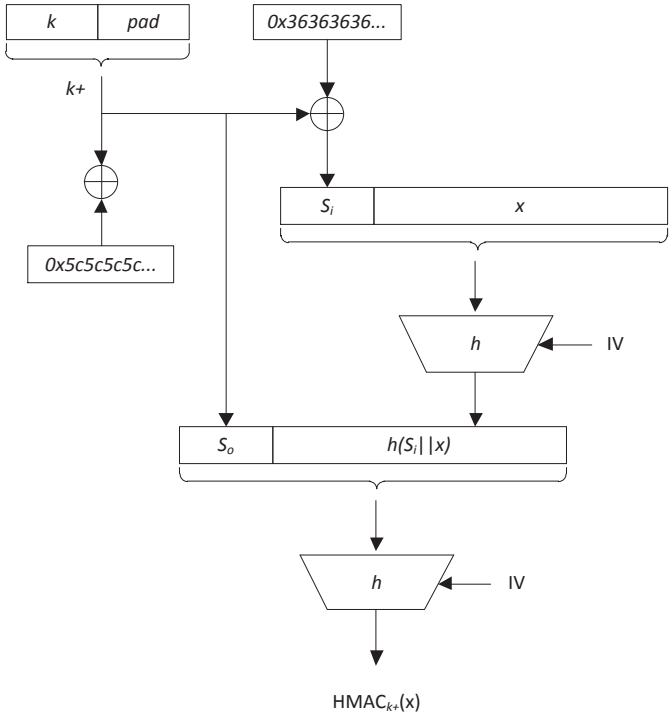

FIGURE 2.10 – Construction HMAC

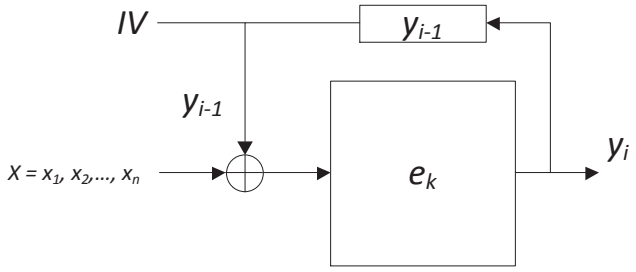

FIGURE 2.11 – Construction CBC-MAC

2.4.1.2 Bloc de chiffrement

Dans le cas des fonctions de hachage construites avec des blocs de chiffrement, le message x est divisé en x_i blocs de taille fixe. Les morceaux du mes-

Attaques récentes sur le mode GCM : Deux publications très récentes (Saarinen, 2011b) (Saarinen, 2011a) (en plus de ceux de Ferguson (Ferguson, 2005)) montrent que le mode GCM possède une classe de clés faibles où celui-ci ne maintient pas toutes les conditions de sécurité. Ils proposent l'utilisation d'un nouveau mode dit chiffrement Sophie-Germain par compteur (Sophie-Germain Counter Mode, SGCM) pour palier à cette faiblesse. Dans un échange privilégié, D. McGrew, inventeur du mode, explique très clairement : "Saarinen (2011b) décrit une façon particulière de falsifier un message, étant donné un message valide, qui fonctionne avec une probabilité d'environ $n/2^{128}$ pour des messages étant de longueur $n \times 128$ bits.". L'observation faite par les auteurs correspond donc à l'analyse de sécurité originale issue de la publication (McGrew et Viega, 2004b). D.McGrew statut par la suite sur l'aspect optimal de cette attaque dans (Saarinen, 2011a) qui, selon ses propres mots : "est intéressante, mais il ne s'agit pas de la description d'une attaque sur le mode GCM fonctionnant avec une chance supérieure de succès que celle décrite précédemment.". "Je ne pense pas que le nouveau mode proposé soit une bonne idée parce qu'il partage la même propriété indésirable du mode GCM à savoir qu'une mauvaise implémentation qui répète les IVs exposera la clé d'authentification.".

HAIFA : HAIFA est suggéré pour remplacer la construction Merkle-Damgård. Il maintient les bonnes propriétés de cette construction en sécurisant la transformation elle même et sa mise à l'échelle. HAIFA possède des propriétés séduisantes : la simplicité, le maintient de la résistance aux collisions des fonctions de compression et une sécurité accrue pour les fonctions de hachage itératives contre les attaques pré-image et pré-image deuxième. HAIFA supporte également des tailles de hachés variable et le hachage aléatoire. HAIFA possède également la propriété en-ligne de la construction Merkle-Damgård. Le calcul d'une fonction de hachage HAIFA requiert une passe sur le message sans conserver le message complet en mémoires.

sage x_i sont chiffrés avec un bloc de chiffrement e de taille de bloc b. Comme m bits de clé sont utilisés, nous utilisons un mapping g de la sortie précédente H_{i-1}, qui est un mapping b vers m bits. Après chiffrement du bloc de message x_i, l'on effectue un ou-exclusif entre le résultat et le bloc original du message. La dernière sortie calculée est le haché du message complet $x_1, x_2, ..., x_n$. La Figure 2.9 montre les 3 constructions typiques pour le hachage basé sur des blocs de chiffrement : la construction Matyas-Meyer-Oseas, Davies-Meyer et Miyaguchi-Preneel. Ces schémas ont pour point commun d'avoir une taille de sortie similaire et égale à la largeur du bloc de chiffrement. Ces fonctions peuvent être exprimées formellement et respectivement par les équations suivantes :

$$H_i = e_{g(H_{i-1})}(x_i) \oplus x_i \tag{2.26}$$
$$H_i = H_{i-1} \oplus (H_i - 1) \tag{2.27}$$
$$H_i = H_{i-1} \oplus x_i \oplus e_{g(H_{i-1})}(x_i) \tag{2.28}$$

2.5 Code d'authentification de message MAC

Les MACs, appelés également fonctions de hachage avec clé ou somme cryptographique, partagent les mêmes propriétés que les signatures numériques. Ils sont cependant basés sur des clés symétriques et par conséquent, ne fournissent pas le service de non répudiation. Un avantage important des MACs est que leur temps de génération et de vérification est comparativement plus faible que les signatures numériques, puisqu'ils sont basés soit sur

UBI : La construction UBI est une variante de la construction cascade (ou de Merkle-Damgård). Elle utilise un bloc de chiffrement tweakable dans le mode Matyas-Meyer-Oseas pour former une fonction de compression et utilise le bit de décalage du bloc étant haché comme tweak.

Le tweak : M. Liskov, R. Rivest, et D. Wagner (Liskov *et al.*, 2002) ont décrit une version généralisée des blocs de chiffrement, appelés blocs de chiffrement "tweakable". Un tel bloc accepte avec l'entrée classique du texte clair ou du chiffré, une deuxième entrée appelée le "tweak". Le tweak, avec la clé, choisit la permutation calculée par le chiffrement.

des blocs de chiffrement, soit sur des fonctions de hachage (MD5, SHA-1).

2.5.1 HMAC

Proposé par Mihir Bellare, Ran Canetti et Hugo Krawczyk, le code d'authentification de message basé sur le hachage (Hash-based Message Authentication Code, HMAC (Krawczyk *et al.*, 1997a)) est une construction qui produit des codes d'authentification de messages basés sur le hachage. Le schéma en Figure 2.10 montre cette construction basée sur deux hachages, l'un extérieur et l'autre intérieur.

Le premier avantage que l'on peut en tirer est la faible pénalité de calcul. Il est à noter qu'un message pouvant être très long n'est haché qu'une seule fois par la fonction de hachage intérieure. L'un de ses autres avantages est le fait qu'il existe une preuve de sécurité qui est fondée sur la sécurité de la fonction de hachage sous-jacente (MD5, SHA-1, ...). Trouver une faille sur HMAC demande que même sans la clé secrète, un adversaire peut construire des tags valides d'authentification de message. Casser la fonction de hachage implique de pouvoir trouver des collisions ou calculer la sortie de cette même fonction sans la connaissance de la valeur initiale IV.

2.5.2 CBC-MAC

Si, comme vu précédemment, les fonctions de hachage peuvent réaliser le calcul de MACs, c'est une possibilité d'utiliser des blocs de chiffrement pour effectuer cette tâche. La méthode la plus commune se dérive de l'utilisation du bloc de chiffrement AES (qui remplaça l'historique DES) dans le mode CBC et est appelé CBC-CMAC. La Figure 2.11 montre le fonctionnement du schéma en question où l'on constate que la taille de sortie du MAC est dépendante de la taille du bloc de chiffrement, i.e. une taille de 128 bits pour AES. Une variation très connue de CBC-MAC est nommé CMAC et est définit dans (Song *et al.*, 2006).

2.5.3 GMAC

Le code d'authentification de message basé sur l'arithmétique de Galois (Galois MAC, GMAC) est issu du mode GCM et est spécifié dans (Nat, 2007). A l'instar de GCM, GMAC est employé uniquement pour calculer

des codes d'authentification de messages et est donc également adapté pour vérifier l'intégrité d'un message. L'utilisation de GMAC par l'encapsulation sécurisé de charge utilie IPSec (Encapsulation Security Payload, ESP) et par l'en-tête d'authentification (Authentication Header, AH) est décrit dans le document RFC 4543 (McGrew et Viega, 2006). GMAC qui peut être implémenté efficacement en logiciel et en matériel, est parallélisable ce qui en fait un excellent candidat concernant les applications haut-débit.

2.6 Evaluation et discussion

Pour évaluer les primitives et modes que nous avons jusqu'alors développés, nous suggérons le Tableau 2.2. Chacune des 5 colonnes représente soit un mode de chiffrement seul, soit un mode de chiffrement-authentifié ou l'authentification est effectuée à l'aide d'une construction par bloc de chiffrement où d'une fonction de hachage dédiée. 15 lignes composent le tableau et précisent les caractéristiques des modes et établissent les pour et les contres associés :

- **Approuvé FIPS** : signifie que l'algorithme ou la technique est décrit soit dans un document de type FIPS ou de type publication spéciale NIST (Special Publication, SP). Ces papiers dressent les différents schémas qui sont approuvés pour une utilisation dont les produits sont développés par des agences appartenant au gouvernement Américain. Ceux-ci ne concerne nullement les agences qui ont un lien proche avec la sécurité nationale qui développent leurs propres standards classifiés. Un algorithme "approuvé FIPS" constitue un sceau d'approbation et, est une caractéristique importante du schéma en question. Dans le contexte des systèmes embarqués sécurisés, les systèmes utilisant exclusivement des techniques approuvées FIPS ont l'avantage d'être de confiance et commercialisable à des agences du gouvernement fédéral. Des techniques ici analysées, toutes sont approuvées FIPS. Aucune vulnérabilité perçue ou documentée n'est à ce jour connue.

- **Prouvé sécurisé** : le mode d'opération authentifié d'un chiffrement est prouvé sécurisé si il peut être prouvé comme étant résistant contre les contrefaçons existentielles sous attaques texte clair choisies. Cela signifie que même si un adversaire à accès a un oracle qui possède la

clé secrète et génère des MACs pour des messages choisis par l'adversaire lui même, celui-ci ne peut déterminer (sans l'aide de l'oracle) le condensa pour tout autre message sans effectuer un nombre de calcul prohibitif. Le mode chiffrement d'un algorithme est typiquement prouvé sécurisé si la sortie du chiffrement ne peut être distinguée d'une chaîne d'un même nombre aléatoire de bits sans effectuer un nombre de calcul prohibitif. La définition exacte de la sécurité prouvée, et par conséquent des hypothèses et bornes utilisées pour les démonstrations de preuve, peut varier. C'est alors que deux schémas prouvés sécurisés peuvent avoir différents niveaux de sécurité. Cependant, en général, les schémas prouvés sécurisés sont préférés. En effet, leur sécurité peut être réduite à des problèmes simples et connus. Tous les modes étudiés ici sont considérés prouvés sécurisés.

- **Nombre de clés AES** : se réfère aux nombres de clés utilisées pour le schéma en question. Ce nombre est égal à deux pour les schémas composés de deux modes indépendants : un pour le chiffrement (e.g., Mode Compteur, CTR), et un second pour l'authentification (e.g., CMAC, GMAC, HMAC). Le nombre de clés AES est égal à un pour les modes conçus pour effectuer du chiffrement et de l'authentification comme CCM ou GCM. Le nombre de clé affecte la quantité de stockage sécurisé qui doit être disponible sur FPGA et peut avoir également une influence sur le temps requis pour l'échange de clé. Dans le cas de notre application spécifique, la différence est minime.

- **Nombre de bits à stocker/transmettre** : se réfère au nombre total de bits de la sortie du chiffrement et de l'authentification. Ce nombre est significatif puisqu'il affecte le temps de transmission dans le cas de téléchargement montant distant, et la taille, et coût de la mémoire flash utilisée pour stocker les applications chiffrées. Si l'on admet un niveau de sécurité similaire pour l'authentification, tous les schémas proposés ici offrent une taille de sortie identique.

- **Temps d'exécution** : Les valeurs des quatre champs suivant affectent le temps total d'exécution des schémas.

  - **Nombre d'appels au bloc de chiffrement** : liés au nombre total de chiffrement et de déchiffrement qui doit être effectué de chaque

côté (i.e., pendant la génération des sorties chiffrées et authentifiées du côté du développeur d'application, et durant le déchiffrement et la vérification des applications sécurisées du côté du système embarqué sécurisé).

- **Appels séquentiels au bloc de chiffrement** : hormis tous les appels au bloc de chiffrement, seul un certain nombre de champs doivent être effectués de manière séquentielle. i.e, la sortie d'opération du premier bloc de chiffrement affecte une entrée d'opération du bloc de chiffrement suivant. Le nombre restant d'appels au bloc de chiffrement (typiquement le cas pour le mode d'opération en compteur, CTR) peut être, en principe, effectué en parallèle.

- **Autre opération séquentielle** : souligne les opérations qui diffèrent des appels au bloc de chiffrement qui sont aussi utilisées pour l'authentification. Des exemples de ce type d'opération correspondent aux appels aux fonctions de hachage pour HMAC et les multiplications dans les corps de Galois pour GMAC et GCM. Selon la vitesse relative du chiffrement et des autres opérations supportées, l'une ou les autres peuvent dominer sur le temps total d'exécution.

- **Nombre d'appel à l'ordonnancement de clé** : le nombre de clés utilisées dans un schéma, affecte le nombre total d'exécution de la routine d'ordonnancement de clé. Cette routine calcule les clés de ronde et se base sur la connaissance de la clé principale. Seul le schéma CMAC combiné avec CTR nécessite deux appels à cette routine.

- **Matériel en amont du chiffrement AES** : facteur important dans un système embarqué où les ressources sont rares et où l'addition de matériel peut substantiellement augmenter le coût total du système. Ce critère favorise les schémas qui nécessitent un minimum de logique supplémentaire comparé à la logique de chiffrement de l'AES : CMAC combiné avec le mode CTR, et le mode CCM.

- **Ordre des opérations** : se réfère à l'ordre dans lequel chiffrement/déchiffrement et génération/vérification de condensa sont effectués, côté émetteur et récepteur. Dans notre cas d'application, il est avantageux

d'avoir la possibilité d'effectuer d'abord l'authentification côté récepteur. Ceci parce que si l'authentification échoue il n'y a pas lieu de passer du temps pour effectuer le déchiffrement. Pour les schémas basés sur deux modes indépendants (l'un pour l'authentification et l'autre pour la confidentialité, l'ordre d'opération est arbitraire. GCM, dans ce cas, a l'avantage de l'ordre d'opération comparé à CCM.

- **Données additionnelles authentifiées** : correspond aux données qui peuvent être non chiffrées, mais doivent être authentifiées. Être dans la possibilité de supporter ce type de donnée est une propriété utile du schéma chiffrement/authentification.

- **"En ligne"** : cette propriété est telle que le traitement de la charge utile peut commencer avant que sa taille complète soit connue. Seul CCM nécessite de connaître la taille de la charge utile à l'avance.

- **Pré-calcul** : désigne la possibilité de commencer des calculs avant que le texte en clair (charge utile) ou le texte chiffré soit disponible. En mode compteur, la majorité des calculs peut être effectuée en avance.

- **IV/nonces et exigence pour les IVs et nonces** : l'utilisation de vecteurs d'initialisations et de nombre utilisé une seul fois "nonces" est très similaire pour chacun des modes. Ce nombre est typiquement obtenu par un compteur (maintenu par l'émetteur) ou une valeur aléatoire (choisie par l'émetteur). La sécurité est maintenue même si l'adversaire peut contrôler ce nombre, sous la contrainte que celui-ci ne peut être répété pendant la session courante (i.e., pendant la période d'utilisation de la clé de chiffrement actuelle). Le nombre n'est pas nécessairement aléatoire, non prédictible, ou secret. Le vecteur d'initialisation, utilisé en mode compteur, possède une exigence de sécurité accrue. Sa valeur doit garantir l'unicité de toutes les valeurs de compteur pour tous les messages chiffrés avec une clé donnée. Alors, non seulement le vecteur d'initialisation mais également les valeurs de compteur suivant ne peuvent être répétés pendant le traitement de deux messages quelconque avec la clé donnée.

Pour résumer, les caractéristiques suivantes des différents schémas semblent être les plus importants du point de vue de notre application :

- Sécurité (garantir que le schéma est approuvé FIPS et prouvé sécurisé).
- Pénalité en surface minimale.
- Latence minimale ou non excessive.
- Exigences minimales de stockage mémoire (taille minimale de la donnée chiffrée et authentifiée).
- Capacité à effectuer l'authentification avec le déchiffrement.
- Support des données additionnelles authentifiées (i.e., authentifié seulement, non chiffrée).

Prenant en considération ces critères, les candidats remarquables sont :

- CMAC + CTR (tous critères exceptés la pénalité en latence).
- CCM (tous critères exceptés la pénalité en latence et l'authentification avant le déchiffrement).
- GCM (tous critères exceptés la pénalité en surface).

Ce que nous devons en tirer, est que la combinaison CMAC + CTR et le mode CCM sont les plus intéressants pour des implémentations qui demandent d'être optimisées en surface et où la latence n'est pas un facteur déterminant. En effet, mis à part la classique "glue" logique nécessaire au contrôle, l'approche ne demande aucun élément de calcul autre que la primitive de chiffrement pour effectuer l'authentification de données. Le coeur de chiffrement qui gouverne la surface globale de l'implémentation, effectue un chiffrement en une dizaine de cycles, chiffre similaire pour l'authentification. Le mode GCM, quant à lui, offre une pénalité en latence très faible, de l'ordre du cycle pour l'authentification, mais la surface s'en retrouve foncièrement agrandi. Ceci est dû au fait que le circuit d'authentification est dissocié de celui du chiffrement, et est dédié à cette tâche.

Le compromis est donc très clair concernant notre application. Pour résoudre la problématique que pose la sécurité haut-débit adaptée aux systèmes embarqués, deux directions centrales peuvent être envisagées. Soit réduire la latence provoqué par l'AES dans le cas des modes CMAC + CTR et CCM et pour cela les implémentations pipelinées de l'algorithme AES sont adéquates mais demandent une surface logique très importante (Ramu et Ananth). Soit réduire la surface de l'élément de calcul d'authentification du mode GCM et c'est ce que nous proposons dans le chapitre suivant.

2.7 Conclusion

Contrairement aux idées communément reçues, le choix des services et des primitives de sécurité est une véritable opportunité pour proposer des services de sécurité adaptés. Savoir répondre précisément à des besoins applicatifs n'est pas l'apanage d'experts en cryptographie, pourvu que l'on comprenne les enjeux qui y sont liés. De bonnes évaluations, notamment équitables, découleront des conceptions plus robustes pouvant mener à des gains économiques importants.

	CMAC, CTR	GMAC, CTR	HMAC, CTR	CCM	GCM
Approuvé FIPS	SP 800-38B, SP 800-38A	SP 800-38Dn SP 800-38A	FIPS 198-1, SP 800-38A	SP 800-38C	SP 800-38D
Prouvé sécurisé	Oui	Oui	Oui	Oui	Oui
Nombre de clés AES	2	2	2	1	1
Nombre de bits à stocker/ transmettre	IVLen + PLen + TLen	IVLen + PLen + TLen	IVLen + PLen + TLen	128 + PLen + TLen	IVLen + PLen + TLen
Nombre d'appels au bloc de chiffrement	2n	n + 2	n	2n + 2	n + 2
Appels séquentielles au bloc de chiffrement	n	2	0	n + 1	2
Autres opérations séquentielles	-	GHASH	HMAC	-	GHASH
Nombre d'appels à l'ordonnancement de clé	2	1	1	1	1
Matériel en	-	multiplication	SHA-2	génération de	multiplication

amont du chiffrement AES		$GF(2^{128})$		compteurs	$GF(2^{128})$
Ordre des opérations	arbitraire	arbitraire	arbitraire	E : Auth, Encr R : Decr, Auth	E : Auth, Encr R : Decr, Auth
Données additionnelles authentifiées	Oui	Oui	Oui	Oui	Oui
"En ligne"	Oui	Oui	Oui	Non	Oui
Pré-calcul	CTR	CTR	CTR	CTR	CTR
IV/nonces, etc.	IV pour CTR	IV pour CTR, IV pour GMAC	IV pour CTR	N	IV
Exigence pour les IVs et nonces	Unicité de chaque compteur	CTR : Unicité de chaque compteur GMAC : Différent pour chaque charge utile	Unicité de chaque compteur	Différent pour chaque charge utile	Différent pour chaque charge utile

Tableau 2.2 – Caractéristiques des modes de chiffrement

Notations

IVLen - Le nombre de bits du vecteur d'initialisation
PLen - Le nombre de bits de la charge utile i.e. le texte clair
TLen - Le nombre de bits du condensa d'authentification
n - Le nombre de blocs de 128 bits de la charge utile (=⌈ PLen/128 ⌉)
GHASH - La fonction d'authentification des modes GCM et GMAC
HMAC - La fonction code d'authentification d'une empreinte cryptographique de message avec clé, spécifié dans FIPS 198-1
SHA-2 - Une des fonction de hachage spécifiée dans FIPS 180-3, autre que SHA-1, i.e., SHA-224, SHA-256, SHA-384 et SHA-512
GF(2^{128}) - Le corps de Galois avec le nombre d'éléments 2^{128}
E : Op1, Op2 - Opérations effectuées du côté émetteur
R : Op1, Op2 - Opérations effectuées du côté récepteur
CTR - Mode compteur d'AES
CCM - Mode compteur avec CBC-MAC
L, R - Les nonces et variable étant dépendantes de la clé
IV - Le vecteur d'initialisation
N - Le nonce

Le cas de la PS3 : Un exemple très concret de mauvaise utilisation de la cryptographie est le cas intéressant de Sony avec sa console de jeu PS3. Tout code effectue une requête au matériel et demande une signature valide pour autoriser son exécution. Dans le cas des exécutables Sony, une signature avec l'algorithme sur les courbes elliptiques ECDSA (Johnson *et al.*, 2001) de l'en-tête est faite avec la clé maître. L'erreur cruciale de Sony vient de l'implémentation de cet algorithme qui nécessite que toutes les signatures soient calculées avec une valeur unique d'un paramètre aléatoire nommé k. Au lieu de cela, Sony utilise une valeur fixe de k pour toutes les signatures des applications, ce qui rend l'algorithme inutile. En effet, dans le cas de ECDSA, lorsque la valeur aléatoire k est constante pour plusieurs générations de signatures, la fonction de hachage peut être résolue pour une clé privé d, de la forme $d = (s \times k - z)/z$ ou s, z et r sont des valeurs publiques. Avec la clé privée d connue, des exécutables peuvent passer la validation de sécurité sans restriction.

Chapitre 3

GHASH comme Fonction d'Authentification

Simplement parce que les applications nécessitant de très haut-débit de production et de consommation de données augmentent, l'authentification cryptographique multi-gigabit, indissociable du chiffrement de message, est un besoin grandissant pour les systèmes embarqués basés sur des FPGA. Dans ce chapitre, nous décrivons pour la première fois une fonction de hachage cryptographique utilisée pour la génération de MACs qui est adaptée aux systèmes embarqués et pouvant produire des débits au-delà des chiffres issus de l'état de l'art.

Le chapitre est organisé de la façon suivante. Le contexte sur l'authentification multi-gigabit et les fonctions de hachage par arithmétique de Galois (Galois HASHing, GHASH) sont donnés en première partie. Les détails d'implémentation de notre approche sont fournis en seconde partie et les résultats expérimentaux sont discutés dans la partie trois, et la quatrième partie donne un aperçu des applications potentielles. La cinquième exprime des directions potentielles pour de futurs travaux et la partie six donne les conclusions.

Sommaire

3.1 Authentification multi-gigabit

Comme l'espace d'application des systèmes FPGA ne cesse de se diversifier, l'importance de solutions efficaces en surface et fournissant de hautes performances a grandi en importance. Par exemple, les communications embarquées haut-débit peuvent maintenant soutenir des débits compris entre 1 et 100 Gbit/s et les communications point-à-périphérique (Intel, 2011) et les transferts de mémoire de masse par lien série avancé (Serial Advanced Technology Attachment, SATA (SATA-IO, 2009)) demandent des bandes passantes brutes d'ordre similaire. Bien souvent, la nature distribuée de ces canaux les rend vulnérables aux attaques ce qui nécessite des mesures de prévention avec de faibles pénalités. De nouveaux blocs de sécurité "optimisés FPGA" et dédiés pour l'authentification de message sont alors nécessaires pour atteindre ce but.

Récemment, le bloc de chiffrement AES dans le mode CTR a été combiné avec le mode d'opération GCM (McGrew et Viega, 2005) pour fournir à la fois du chiffrement et de l'authentification. Cette approche est populaire puisqu'elle n'est pas contrainte par les droits sur la propriété intellectuelle et est prouvée sécurisé (Dworkin, 2007). L'aspect clé de GCM est la multiplication 128 bits dans le corps de Galois GF(2^{128}). Une ou plusieurs instantiations de cette opération GMULT est nécessaire pour effectuer la fonction de hachage dite de Galois (GHASH) et utilisée pour l'authentification de message. Mathématiquement, une fonction cryptographique GHASH est une construction qui effectue du hachage universel sur un corps de Galois binaire pour générer le MAC (Wegman et Carter, 1981). Le but de cette fonction est d'authentifier la source d'un message et son intégrité. Bien que des implémentations matérielles et logicielles de GHASH sont disponibles (Wang *et al.*, 2010), la plupart requiert l'utilisation d'élément de multiplication qui ne sont pas adaptés pour des FPGAs modestes. Dans ce travail, une implémentation optimisée pour un déploiement efficace sur FPGA est décrit. Le module GHASH est construit pour prendre avantage de la structure table de scrutation (LookUp Table, LUT) des FPGA et de leurs registres disponibles (Flip-flop, FF). La clé personnalisée utilisée pour l'authentification est synthétisée dans la structure du module, ce qui spécialise le circuit associé et minimise sa surface. Une version pipelinée du module est présentée pour fournir de hauts débits. Globalement, le module est spécialement optimisé pour cibler des FPGAs faible coût possédant peu de logique qui sont typiquement choisis pour des appli-

cations embarquées, comme la télécommunication, avec de fortes contraintes en débit de données. Des débits de plus de 292 Gbit/s ont été évalués.

3.1.1 Description fonctionnelle de la fonction GHASH

Comme montré en Figure 3.1, la fonction GHASH est composée d'éléments chaînés de multiplications dans GF(2^{128}) (GMULT) et d'opérations ou-exclusif (eXclusive OR, XOR). Les entrées de la fonction inclues :

- Une clé de hachage 128 bit H. Cette clé est dérivée d'une clé cryptographique de chiffrement symétrique K.

- D'un message de M-bits qui doit être authentifié. Le message peut être divisé en n blocs de 128 bits $M_1 - M_n$. Si nécessaire, le dernier bloc du message M_n est complété avec des zéros pour former un mot de 128 bits.

- Un mot optionnel de 128 bits de données à authentifier (ADD). Cette valeur de donnée, qui est authentifiée mais pas chiffrée, est généralement utilisée pour identifier la source d'un message.

- Une valeur de 128 bits LEN qui exprime la taille des mots AAD et du message M.

- Un masque jetable cryptographique de 128 bits (PAD) qui chiffre la sortie TAG de la fonction GHASH pour générer le MAC.

Le résultat de 128 bits est exprimé par les équations suivantes :

$$
\begin{aligned}
H &= E(K, 0^{128}) && (3.1)\\
X_0 &= GMULT(H, AAD) && (3.2)\\
X_i &= GMULT(H, M_i \oplus X_{i-1}) && (3.3)\\
&\quad pour\ i = 1...n &&\\
LEN &= length(AAD)_{64} \,||\, length(M)_{64} && (3.4)\\
TAG &= GMULT(H, X_n \oplus LEN) && (3.5)\\
MAC &= PAD \oplus TAG && (3.6)
\end{aligned}
$$

Où E(K,B) dénote le chiffrement AES de la valeur B avec la clé secrète K. L'expression 0^{128} dénote une chaîne de 128 bits à zéro, et A‖B dénote la concaténation des chaînes de bits A et B. La multiplication de deux éléments A,B∈GF(2^{128}) est notée GMULT(A,B), et l'addition dans le corps de Galois de A et B, dénotée A⊕B, est équivalente à une opération XOR. La fonction length() retourne un mot de 64 bits décrivant le nombre de bits de son argument, avec le bit le moins significatif à droite. En général, une implémentation efficace de GHASH dépend de la conception logicielle ou matérielle de l'élément de multiplication dans GF(2^{128}).

3.1.2 Implémentations matérielles de GHASH

Précédemment, Paar (Paar, 1999) a résumé l'efficacité de la multiplication matérielle dans les corps finis GF(2^q). Bien que les implémentations bit-série ont une surface linéaire avec une performance en $O(q)$ et les implémentations digit-série varient selon la taille du digit D pour une surface en $O(qD)$ et une performance en $O(q/D)$, leurs performances sont généralement considérées comme insuffisantes pour un débit d'authentification de message contemporain.

Même si les tailles des implémentations parallèles sont généralement supérieures aux implémentations séries, le désir d'obtenir les meilleures performances en débit requiert leur utilisation. Comme la complexité d'une implémentation parallèle est en $O(q^2)$, pour une implémentation 128 bits $O(q^{128})$ plus de 10 000 LUTs peuvent être aisément nécessaires si des optimisations matérielles ne sont pas considérées. Fort heureusement, la multiplication dans GF(2^{128}) peut être exprimée par une série de multiplications polynomiales et des réductions modulaires menant aux implémentations basées sur les algorithmes Reyhani-Masoleh (Huo *et al.*, 2009), Mastrovito (Wang *et al.*, 2010) et Karatsuba-Ofman. Ces implémentations montrent qu'il est possible d'atteindre des débits de l'ordre du multi-Gbit/s à un coût supérieur à 8000 LUTs par fonction. En section 3.5, une comparaison de notre nouvelle implémentation de GHASH qui inclus l'élément de multiplication GF(2^{128}) est effectuée par rapport à ces approches.

3.1.3 Implémentations logicielles de GHASH

Les idées pour une implémentation efficace de la multiplication dans GF pour la fonction GHASH peuvent être identifiées en considérant les implémentations logicielles précédentes. La multiplication dans un corps binaire en logiciel utilise généralement une variété de compromis latence-surface (Shoup, 1996). Actuellement, les implémentations logicielles prennent deux formes, une qui considère la clé de hachage H de l'équation (3.1) comme fixe et l'autre qui considère une valeur de H pouvant varier au cours du temps. L'opération de multiplication GMULT(H,B) entre la clé de hachage H et un élément arbitraire B, comme montré dans les équations (3.2), (3.3) et (3.5), est linéaire dans le corps GF(2). En admettant la clé H constante, cette propriété peut être exploitée pour produire des résultats efficaces en se fondant sur une approche "dirigée-table" (Shoup, 1996). Dans beaucoup de cas, cette approche fournit des performances significatives en logiciel pour un coût modeste en mémoire. L'implémentation par table peut être optimisée pour limiter la mémoire nécessaire pour encoder les opérations basées sur H et autoriser un accès parallèle dans le but d'obtenir de hauts débits.

3.1.4 Aperçu de la nouvelle approche matérielle

Pour l'implémentation de notre module matériel optimisé en surface, les optimisations pour une clé constante sont adaptées pour une architecture FPGA. Le parallélisme grain fin trouvé dans les FPGAs est utilisé pour implémenter une table pré-calculée pour les opérations GMULT basée sur une valeur de H constante. Comme montré en Section 3.3, ceci fournit une implémentation d'un élément parallèle de multiplication dans GF(2^{128}) avec un nombre de LUTs significativement réduit tout en fournissant un débit de plusieurs Gbit/s. Les bénéfices spécifiques de cette implémentation de GHASH inclues :

1. Un gain en surface important par la réduction de la surface de la fonction GHASH de 50% en l'optimisant pour une implémentation FPGA. Ceci autorise un déploiement efficace sur FPGAs faible coût communément utilisé dans les applications embarquées.

2. La spécialisation de la table inclus dans GMULT peut être accommodée pour de multiples clés si un portfolio de bitstreams pour différentes

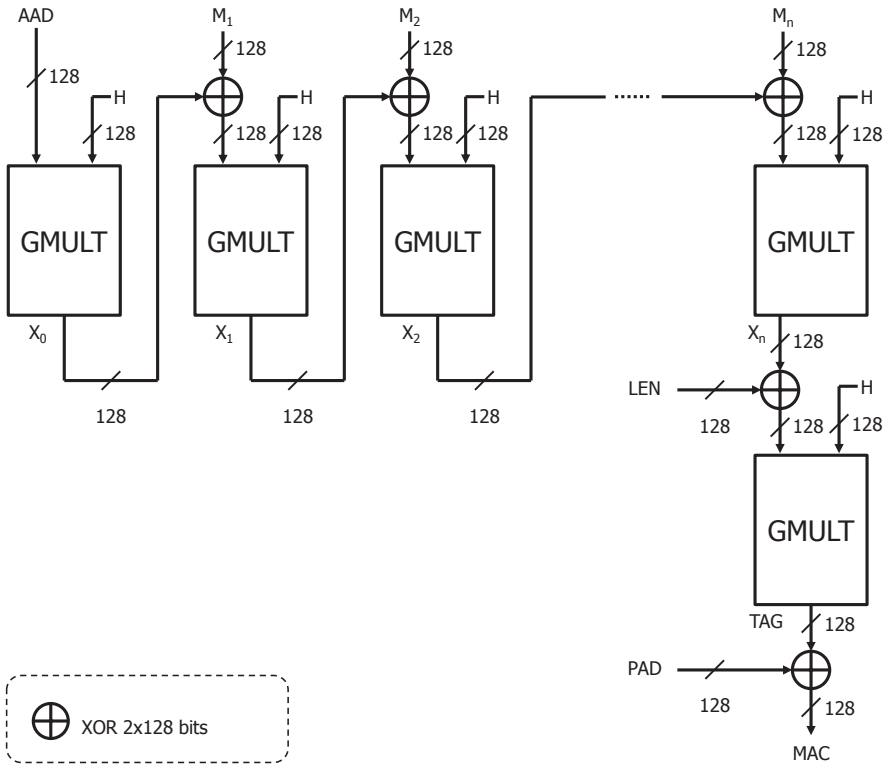

FIGURE 3.1 – Diagramme bloc de l'implémentation combinatoire de la fonction GHASH

valeurs de H est maintenu.

3. L'implémentation est suffisamment petite pour que de multiples instantiations avec des clés isolées puissent être utilisées pour gérer plusieurs flux de données.

Les détails spécifiques de l'implémentation sont décrit dans la section suivante.

3.2 Architectures du module GHASH

Dans cette section, un module de base GHASH qui génère une donnée d'authentification de 128 bits tous les deux cycles d'horloge est décrit. La conception combine deux blocs combinatoires GMULT montrés à gauche de la Figure 3.1 avec un registre de sortie. Ce modèle est conçu pour être aisément intégré dans une conception complexe. De multiples instantiations de ce module peuvent être chaînées ensemble pour former une implémentation pipelinée supportant des débits supérieurs.

3.2.1 Pré-calcul de table

L'utilisation efficace d'une table de scrutation GMULT basée sur une valeur H fixe demande le pré-calcul des valeurs de la table. Pour une valeur constante de H, une table T peut être construite pour représenter les multiplications dans GF(2^{128}) entre H et une valeur d'entrée de 128 bits. Chaque élément de la table est un vecteur de 128 bits. Fondé sur les spécifications de GMULT (McGrew et Viega, 2005), l'algorithme nécessaire pour le remplissage de la table est décrit par l'algorithme 1. Le bit i d'un élément A est dénoté A_i. Le bit le plus à gauche est A_0 et le bit le plus à droite A_{127}. L'opération de multiplication utilise l'élément $R = 11100001\|0^{120}$ (McGrew et Viega, 2005). La fonction rightshift() décale le bit de son argument d'un bit à droite. La clé de hachage H est le résultat de l'invocation du bloc de chiffrement AES avec un mot 0 de 128 bits à son entrée. L'opération est soulignée en ligne 2 de l'algorithme 1 ou K est la clé secrète. Chaque vecteur de 128 bits T est calculé avec de simples tests conditionnel avec des XORs 128 bits et des décalages à droite (ligne 5 à 9).

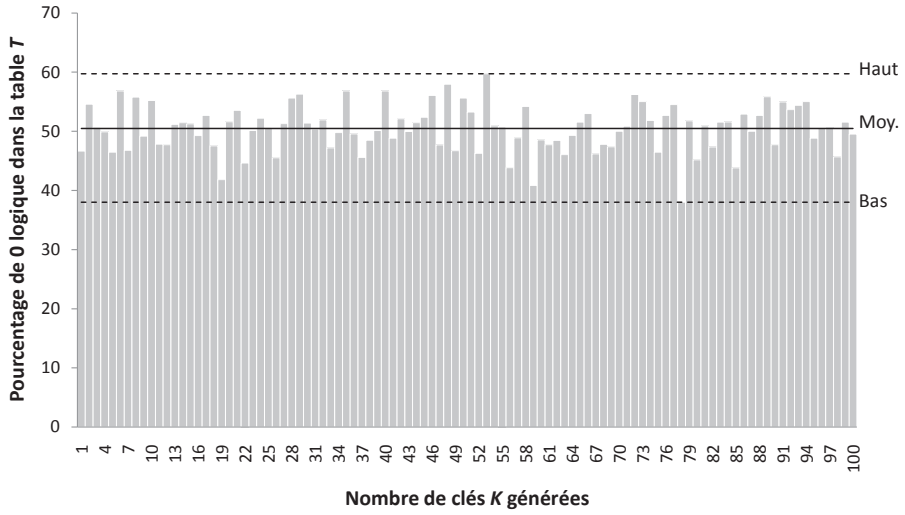

FIGURE 3.2 – Pourcentage de bits à l'état logique 0 pour 100 clés K générées

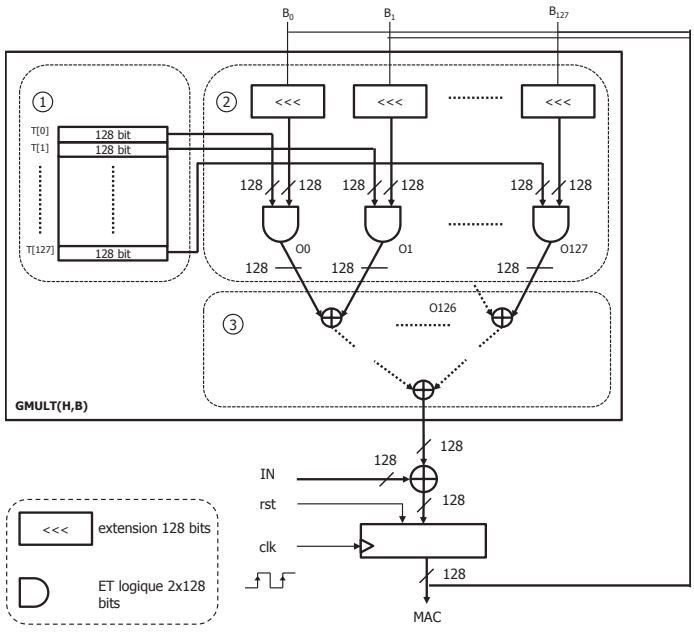

FIGURE 3.3 – Implémentation du module GHASH basic

Algorithme 1 *Pre − calcul de la table* T; $H \in GF(2^{128})$

1: $R \leftarrow 11100001 \parallel 0^{120}$
2: $H \leftarrow E(K, 0^{128})$
3: $V \leftarrow H$
4: **for** $i = 0$ to 127 **do**
5: **if** $V_{127} = 0$ **then**
6: $V \leftarrow rightshift(V)$
7: **else**
8: $V \leftarrow rightshift(V) \oplus R$
9: **end if**
10: $T[i] \leftarrow V$
11: **end for**

La table non optimisée contient 128 vecteurs de 128 bits, i.e. 16384 bits mémoires stockés soit dans des registres soit en mémoire à accès direct (Random Access Memory, RAM). Additionnellement, comme expliqué dans la sous-section suivante, une large fraction de ces bits doit pouvoir être accédée en parallèle. Une implémentation, directe de cette table en LUTs est clairement prohibitif de part la taille et les problèmes de performances inhérentes aux plus de 10000 LUTs nécessaires. Une implémentation directe dans les blocs de mémoire (Block RAM, BRAM) peut sembler une option raisonnable puisque ces primitives opèrent à haute vitesse (e.g. 550 mégahertz (MHz) sur un Virtex-5). Cependant, les BRAMs ont tout au plus 2 ports de lecture, ce qui limite l'accès en parallèle. Pour être capable d'effectuer un accès parallèle à ces données, le contenu du bloc RAM devrait être nécessairement répliqué un nombre important de fois (e.g. plus de 64x pour un dispositif Spartan). En général, beaucoup de conceptions sur FPGA sont contraintes par la disponibilité des BRAM.

Notre implémentation évite le problème de RAM distribuée et parallèle en synthétisant les valeurs binaires 1 de la table T directement dans la logique du bloc GMULT. Pour la plupart des valeurs H, la population des bits de T n'est pas composée strictement de 50% de bits à 1 et 50% de bits à 0, ce qui mène à de possibles optimisations. De ce fait, les sorties des portes montrées en Figure 3.3 peuvent être mises à 0 dans beaucoup de cas, ce qui réduit la logique nécessaire pour implémenter la fonction GHASH. Les entrées des portes logiques de la figure qui sont fixées à 1 convertissent les portes sui-

vantes en conséquence. De ce fait, l'arbre de XOR de dimension 128×128 peut être élagué. En supprimant les seuls bits avec un niveau logique 0 et en optimisant les valeurs logiques 1 avant le processus de mapping, la conception une fois synthétisée est réduite en surface. De plus, la logique est structurée pour prendre avantage de la structure de LUTs du FPGA qui sont à entrée large et simple sortie.

Pour étudier la distribution moyenne des valeurs logiques d'une table T, nous avons aléatoirement généré 10^8 clés K et évalué la population des bits qui ont une valeur logique 0. L'algorithme de génération de nombres pseudo-aléatoires Mersenne Twister (Matsumoto et Nishimura, 1998) a été choisi pour générer les valeurs de K. Cet algorithme a pour avantage de posséder une période très longue 2^{19937}-1, est uniformément distribué, et passe les nombreux tests statistiques sur les propriétés aléatoires tels que les tests Diehard (Marsaglia, 1995) ou les tests TestU01 (L'Ecuyer et Simard, 2007). Pour cette gamme de clés K, des pourcentages faibles de 30%, moyen 50% et haut 70% de valeurs logiques 1 ont été déterminés. Un exemple de distribution pour 100 clés K est montré en Figure 3.2. La densité logique de la conception est en relation immédiate avec cette distribution. Le module le plus petit et le plus rapide est obtenu pour un pourcentage de 30%. Une distribution de 50% résulte en un module plus gros et donc plus lent. Nos conceptions implémentées en Section 3.3 considèrent une distribution de 50%.

3.2.2 Module basic

Le module non optimisé GHASH montré en Figure 3.3 contient un élément de multiplication purement combinatoire dans GF(2^{128}) GMULT(H,B), un XOR 2×128 bits et un registre de sortie 128 bits. Le détail de l'implémentation de cet élément (Algorithme 2) est la base de cette architecture. Trois sous modules forment la base de cette conception :

- La table T 128×128 synthétisée dans la logique (à gauche).
- Le circuit pour effectuer les tests conditionnels (à droite).
- Le XOR 128×128 bits (en bas à droite).

Pour éviter le cycle de pénalité durant les tests conditionnels synchrones, ceux-ci sont implémentés de manière combinatoire. Chaque bit d'entrée B, B_0 à B_{127}, est étendu 128 fois. Un ET logique 2×128 est alors effectué avec le vecteur de la table T correspondant. Le module accepte une seule entrée 128 bits IN et génère une sortie 128 bits MAC. Un chemin de rétroaction permet l'itération de tous les blocs de 128 bits du message à authentifier, comme montré en Figure 3.1. L'opération du module est contrôlé par un petit séquenceur. Par exemple, pour authentifier un message composé de 128 bits d'ADD et d'un message de 512 bits M, l'entrée IN suit la séquence AAD, $M1$, $M2$, $M3$, $M4$, LEN et PAD. Le MAC résultant est disponible et valide dans le registre de sortie après 6 cycles.

3.2.3 Module pipeliné

Comme montré en Figure 3.4, un module pipeliné multi-étages peut être dérivé d'instantiations multiples du module basic. La conception basic est répliquée n fois pour fournir un module pipeliné n-stage. Pour chaque nouvel étage, un GF(2^{128}) GMULT(H,B), un XOR 2×128 bits et un registre 128 bits est ajouté. Les Figures 3.4 et 3.5 décrivent respectivement l'architecture et fait état des chronogrammes pour une conception pipeline à deux étages. Une fois initialisée, cette conception sort un MAC de 128 bits d'AAD et de 128 bits de message M tous les cycles d'horloge. La rétroaction n'est pas nécessaire pour des messages de 128 et 256 bits puisque l'authentification est effectuée tous les cycles. Pour authentifier des messages plus longs que 256 bits, la conception pipeline doit être adaptée en :

- Fournissant un chemin de rétroaction du dernier étage jusqu'à l'entrée du premier étage.

- Chargeant le registre de sortie correspondant avec la valeur ADD si nécessaire.

- Ordonnançant correctement les données. Pour cette étape, l'entrée IN peut recevoir, au choix, un bloc de message 128 bits, une valeur LEN de 128 bits ou une valeur PAD de 128 bits.

Les simulations ont été effectuées sous conditions nominales de tension (0.95V), de température (85°C) et de paramètres d'efforts de mapping et

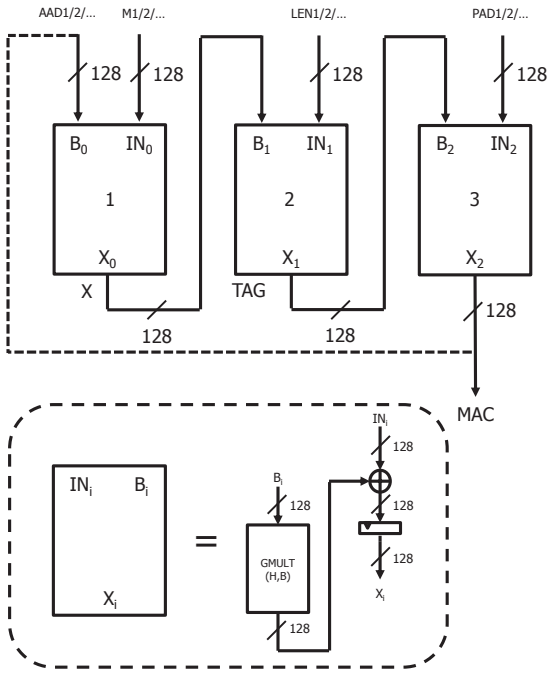

FIGURE 3.4 – Implémentation du module GHASH avec deux étages de pipeline

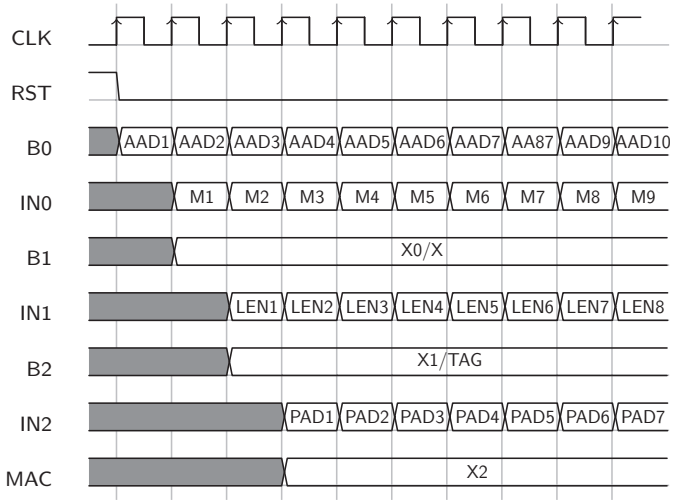

FIGURE 3.5 – Chronogrammes du module GHASH avec deux étages de pipeline

de placement-routage. Les aspects temps relatifs aux entrées/sorties ont été omis pour considérer les modules en tant que fonctions autonomes.

3.3 Résultats

Les deux modules GHASH, basic (Figure 3.3) et pipeline (Figure 3.4) sont décrits en Verilog et ciblent plusieurs familles contemporaines de FPGA Spartan et Virtex avec l'utilisation des outils de synthèse Xilinx Synthesis Technology (Xilinx Synthesis Technology, XST) et la suite logicielle intégrée (Integrated Software Environment, ISE) 13.1. ModelSim 6.6a a été utilisé pour la simulation avant et après placement-routage.

3.3.1 Evaluations des performances

Le Tableau 3.2 fournit un résumé des fréquences, débits et utilisations des ressources pour ces nouvelles conceptions comparées aux résultats précédemment publiés. Le Tableau 3.2 est fourni à titre de référence "haut-niveau" puisque les conceptions précédentes sont les plus similaires par nature à ceux rapportées dans ce chapitre. A l'inverse des efforts précédents, notre nouvelle approche est optimisée pour les LUTs FPGA et spécialisée par clé de hachage. Parce qu'il existe une large diversité d'implémentations précédentes, nous établissons la comparaison avec les conceptions ne contenant qu'un élément de multiplication GF(2^{128}) (M), une architecture GHASH complète (G) et si la structure est clé-dépendante (K). Notez les bulles remplies dans les colonnes M/G/K. De plus, certains résultats de performances rapportés sont donnés avant placement-routage comme indiqué par le tableau (colonne PAR). Dans certains cas, seuls les nombres de LUTs sont donnés. Dans ces cas, le nombre de slices est estimé en considérant le nombre de LUTs et de registres par slice pour l'architecture ciblée. Pour évaluer l'efficacité en surface de notre approche dans le contexte de l'utilisation de ressources, nous utilisons la métrique débit-par-slice pour comparer les architectures. Cette métrique est largement utilisée par la communauté cryptographique.

Si une augmentation du débit est nécessaire dans notre architecture, de multiples copies pipelinées pourraient être instanciées, comme discuté en Section 3.3. Pour de hautes performances, le module basic opère à 384.6 MHz sur un dispositif Virtex-6. La conception utilise 143 FFs ce qui inclue le re-

	Ressources			Fréquence	Débit	
	LUTs	FFs	Slices	(MHz)	(Gbit/s)	(Mbit/slice)
Spartan-3 (xc3s1000l-4fg456)						
Borne supérieure	2403	142	1225	105.4	6.8	5.5
Borne inférieure	1589	136	811	135.2	8.7	10.7
Virtex-5 (xc5vlx50t-3ff1136)						
Borne supérieure	2482	157	810	256.4	16.4	20.2
Borne inférieure	968	136	327	345.1	22.1	67.6
Virtex-6 (xc6vlx75t-3ff484)						
Borne supérieure	1914	142	839	278.1	17.8	21.2
Borne inférieure	963	162	272	418.1	26.8	98.5

Tableau 3.1 – Variabilité des résultats post placement-routage du module basic sur Spartan-6, Virtex-5 et Virtex-6

gistre de sortie 128 bits et des registres dupliqués pour améliorer le fan-out, et 1714 LUTs qui sont principalement utilisées pour l'élément de multiplication GF(2^{128}). Deux cycles de calcul sont nécessaires par sortie, alors un débit de $384.6\times10^6\times128/2$=24.6 Gbit/s est obtenu.

Un module GHASH pipeliné de 8 étages opère à 285.7 MHz sur un Virtex-6. Au total, il consomme 1358 FFs et 15414 LUTs. Cette implémentation produit un MAC 128 bits d'un message de 1 Ko tous les cycles. Un seul cycle de calcul est nécessaire alors $285.7\times10^6\times128\times8$=292.6 Gbit/s de débit est obtenu. Le débit-par-slice est de 73.4 Mbit/slice.

Le Tableau 3.1 montre la variabilité possible des résultats d'implémentation post placement-routage du module basic pour trois FPGAs (Spartan-3, Virtex-5 et Virtex-6) pour deux clés diamétralement opposées : une clé produisant une forte sparsité de la table T (72.4%, borne inférieure de conception), et une clé produisant une faible sparsité (29.9%, borne supérieure). Ces deux clés sont issues d'une recherche exhaustive selon les algorithmes décrits en Section 3.2.

Les codes sources des conceptions matérielles (Hardware Description Language, HDL), fichiers de simulation, et les outils logiciels pour reproduire les résultats pour ces deux variantes de module sont publiquement disponibles à cette URL : `http://code.google.com/p/ghash/`

Algorithme 2 *Multiplication dans* $GF(2^{128})$. *Calcul la valeur* $X = GMULT(H, B)$ *avec* H *constante;* $B, X \in GF(2^{128})$

1: **for** $i = 0$ to 127 **do**
2: $X \leftarrow 0$
3: **if** $B_i = 1$ **then**
4: $X \leftarrow X \oplus T[i]$
5: **end if**
6: **end for**
7: **return** X

Conception	Dispositif				Ressources			Fréquence	Débit	
	Famille	FPGA	M / G / K	PAR	LUT	FF	Slices	(MHz)	(Gbit/s)	(Mbit/slice)
Basic GHASH	Virtex-4	xc4vlx25-12ff668	○ / ● / ●	●	2469	165	1277	250.0	16.0	12.5
Basic GHASH	Virtex-5	xc5vlx50t-3ff1136	○ / ● / ●	●	2008	128	533	303.0	19.4	36.4
Basic GHASH	Virtex-6	xc6vlx75t-3ff484	○ / ● / ●	●	1714	143	447	384.6	24.6	55.1
Basic GHASH	Spartan-3	xc3s1000l-4fg456	○ / ● / ●	●	2429	150	1242	114.9	7.4	5.9
Pipe GHASH (n=4)[1]	Virtex-4	xc4vlx25-12ff668	○ / ● / ●	●	12186	793	6182	222.2	113.8	18.4
Pipe GHASH (n=8)[1]	Virtex-5	xc5vlx50t-3ff1136	○ / ● / ●	●	17611	1152	4594	232.6	238.1	51.8
Pipe GHASH (n=8)[1]	Virtex-6	xc6vlx75t-3ff484	○ / ● / ●	●	15414	1358	3985	285.7	292.6	73.4
Pipe GHASH (n=1)[1]	Spartan-3	xc3s1000l-4fg456	○ / ● / ●	●	4875	280	2484	116.3	14.9	6.0
Huo *et al.*	Virtex-5	xc5vlx30-3ff324	● / ○ / ○	○	8864	411	2992[2]	240.2	30.8	10.3[2]
Lu *et al.*	Virtex-5	xc5vlx50t-3ff1136	○ / ● / ○	○	9405	430	3175[2]	120.2	15.4	4.8[2]
Zhou *et al.*	Virtex-5	xc5vlx85-2ff668	● / ○ / ○	●	4214	2084	4628	298.8	38.2	8.3
Chen *et al.*	Virtex-4	xc4vlx60-11ff668	○ / ● / ○	●	*n/a*	*n/a*	10756	312.5	40.0	3.7
Henzen et Fichtner	Virtex-5	xc5vlx220-2	○ / ● / ○	●	*n/a*	*n/a*	14799	233.0	119.3	8.1
Wang *et al.*	Virtex-5	xc5vlx85t-3ff1136	○ / ● / ○	○	32410	2612	10943[2]	240.3	123.1	11.2[2]

[1] nombre d'étages de pipeline
[2] estimation basée sur le nombre de LUTs par slice

Tableau 3.2 – Comparaison des modules proposés avec les approches précédentes

3.3.2 Discussion

Pour toutes les comparaisons mesurées, notre nouveau module GHASH montre un débit-par-surface amélioré. Par exemple, notre module atteint un facteur 4x d'amélioration en débit-par-surface pour un Virtex-5 comparé au compétiteur le plus proche (Wang *et al.*, 2010). Un débit brut sur Virtex-5 de 238.1 Gbit/s est supérieur aux approches précédentes avec une réduction en surface de l'ordre de 50%. La nature bas-coût de notre conception permet l'implémentation sur des dispositifs bas-coûts type Spartan-3. La version pipeline du module atteint alors 14 Gbit/s de débit et utilise 4875 LUTs compactées dans 2484 slices.

L'implémentation d'une fonction d'authentification cryptographique multi Gbit/s sur des FPGAs faibles coûts est une nouvelle caractéristique de ce travail. A notre connaissance, aucune fonction d'authentification standard n'est connue pour avoir des résultats similaires ou meilleurs sur des FPGAs de taille réduite. Les travaux précédemment ont exploré de façon pléthorique l'espace de conception de l'algorithme SHA. Dans cette optique déjà évoquée de se comparer équitablement avec les travaux existants et récents, nous référençons dans le Tableau 3.3 les résultats d'implémentation des candidats au futur standard SHA-3. Nous soulignons avec insistance et rappelons que l'algorithme SHA et ses variantes permettent la non répudiation, ce qui n'est pas le cas de GHASH. Cependant la construction HMAC fondée sur SHA (et donc dans un futur très proche sur SHA-3) étant très utilisée avec l'algorithme SHA, nous pensons la comparaison appropriée.

Bien que nos résultats sont intéressants en terme de surface réduite et de haut-débit, l'efficacité de notre conception est liée à la fréquence pour laquelle la clé de hachage H, dérivée de la clé secrète K, doit être changée. Selon les spécifications AES-GCM, une seule clé autorise l'authentification de 4 Go/s (32 Gbit/s) de données pour 64 ans sans compromettre la sécurité (McGrew et Viega, 2005). Ce terme est généralement bien plus long que la durée de vie d'un système numérique. Cependant, le changement de clé peut être réclamé plus fréquemment dépendamment de l'application. La nature reconfigurable d'un FPGA rend le changement de clé de hachage qui est embarqué dans le matériel possible. Un nouveau bitstream peut être dynamiquement chargé dans le FPGA, quand nécessaire. Bien que les approches (Henzen et Fichtner, 2010), (Wang *et al.*, 2010), (Huo *et al.*, 2009), (Lu *et al.*, 2009) et (Zhou *et al.*,

2009), ne demandent qu'un seul chargement de registre pour fournir une nouvelle clé, notre approche est valide pour un bon nombre d'applications.

3.4 Applications

3.4.1 Application à la sécurité des mémoires

Les mémoires systèmes hors-puce d'un système embarqué peuvent rencontrer une variété importante d'attaque (Elbaz *et al.*, 2006). Une mémoire hors-puce peut être observée et falsifiée aisément si aucune politique de protection n'est prise en considération. Avec un chiffrement/déchiffrement symétrique comme AES, la protection de mémoire pointe l'utilisation de l'authentification mémoire. Le bus Serial SATA (SATA-IO, 2009) permet une capacité brut de transfert de 6 Gbit/s. En prenant appui sur le Tableau 3.2, il est évident que notre implémentation de GHASH sur Spartan-3 est compatible pour implémenter la partie sécurité du contrôleur pour effectuer l'authentification. L'exemple d'un tel contrôleur est montré en Figure 3.6.

3.4.2 Application aux VPNs

Les attaques réseaux sont une préoccupation pour un bon nombre d'organisations. Prévenir les accès non-autorisés, les modifications, et les mauvais usages et les dénis de service (Denial of Service, DoS) sont des clés pour fournir un environnement sécurisé. Les réseaux privés virtuels (Virtual Private Network, VPN) sont largement employés pour connecter des réseaux locaux (Local Area Network, LAN) à des lieux distants par l'utilisation d'un tunnel sécurisé qui protège le canal de transmission de paquets. Pour beaucoup de ces structures, la clé secrète utilisée pour le chiffrement et l'authentification est changée sur une base journalière, hebdomadaire, mensuelle ou annuelle. Chacune communique avec les autres en utilisant un canal sécurisé point-à-point. La Figure 3.7 montre la connectivité d'un VPN simple ou trois LANs représentent un bureau de quatre personnes. Dans cette configuration les flux de paquets LAN sont authentifiés avec un seul module partagé GHASH basé sur une clé secrète. Une infrastructure plus complexe ou chaque LANs possède une clé différente peut être construite en instanciant un module par flux de paquets.

Algorithme	Taille du bloc (bits)	Fréquence (MHz)	Cycles d'horloges	Débit (Mbps)	Slices	Registres	LUTs
Finalistes							
BLAKE	512	115	22	2676	1660	1393	5154
Grostl	512	154	10	7885	2616	1570	10088
JH	512	201	39	2639	2661	1621	8392
Keccak	1024	205	24	8397	1433	2666	4806
Skein	256	115	21	1402	854	929	2864
Non finalistes							
BMW	512	34	2	8704	4530	1317	15012
CubeHash	256	185	16	2960	590	1316	2182
ECHO	1536	149	99	2312	2827	4198	9885
Fugue	32	78	2	1248	4013	1043	13255
Hamsi	32	210	4	1680	718	841	2499
Luffa	256	261	9	7424	1048	1446	3754
Shabal	512	228	50	2335	1251	2061	4219
SHAvite-3	512	251	38	3382	1063	1363	3564
SIMD	512	75	46	835	3987	6693	13908
SHA-256	512	260	68	1958	609	1224	2045

Tableau 3.3 – Résultats d'implémentation des candidats SHA-3 sur Virtex-5 xc5vlx30

Les appareils de sécurité commerciaux actuels permettent un débit maximal de 40 Gbit/s et autorisent potentiellement plus de 10000 sessions VPN utilisateur (Cis, 2011). Les opérations cryptographiques sous-jacentes pénalisent le débit VPN par un facteur 8 i.e 5 Gbit/s. Particulièrement, l'authentification est basée sur les algorithmes SHA qui sont consommateurs en nombre de cycles. L'infrastructure VPN peut largement bénéficier de l'approche GHASH clé-dépendante pour obtenir un débit proche des 40 Gbit/s atteignable. Un tel gain pousse le nombre d'utilisateurs à augmenter ou à prendre avantage d'une meilleure bande passante. Les VPNs standards mais également mobiles devraient suivre ce besoin grandissant de bandes passantes larges capacités. Ces travaux donnent une réponse préliminaire à cette demande.

Cette section n'est pas exhaustive en termes d'applications possibles. Du chargeur de boot, de code, et du chargement protégé de bitstreams aux communications sécurisées d'unités de calculs hautes performances, l'aspect clé-dépendant de GHASH est compatible. Cependant comme mentionné plus tôt, toutes les applications ne sont pas possibles avec cette approche. Le standard de chiffrement de l'institut des ingénieurs en électricité et en électronique (Institute of Electrical and Electronics Engineers, IEEE) MACSEC Han *et al.* (2006a) par exemple, qui utilise AES-GCM pour le chiffrement et l'authentification (Han *et al.*, 2006b). En utilisation typique, la clé requise peut changer de paquet à paquet. Notre implémentation demanderait dans ce cas précis une reconfiguration du FPGA à chaque nouvelle réception, ce qui est prohibitif.

3.5 Conclusion

Dans ce chapitre, une nouvelle implémentation optimisée FPGA du bloc d'authentification de message GHASH à été présentée. L'implémentation clé-dépendante minimise l'utilisation de la logique pour faciliter son intégration dans des FPGA faible-coût communément utilisés dans les systèmes embarqués. Le module prend avantage de la spécialisation offerte par les FPGAs pour personnaliser le matériel basé sur une clé secrète GHASH. L'architecture de ce module permet une réduction matérielle importante puisque la table associée à GHASH n'est pas uniforme. Globalement, nous obtenons

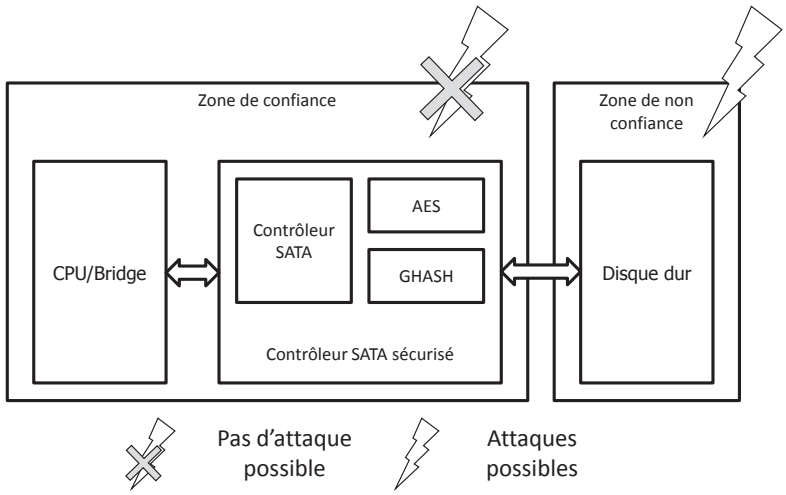

FIGURE 3.6 – Modèle haut niveau d'un contrôleur SATA sécurisé et basé sur GHASH

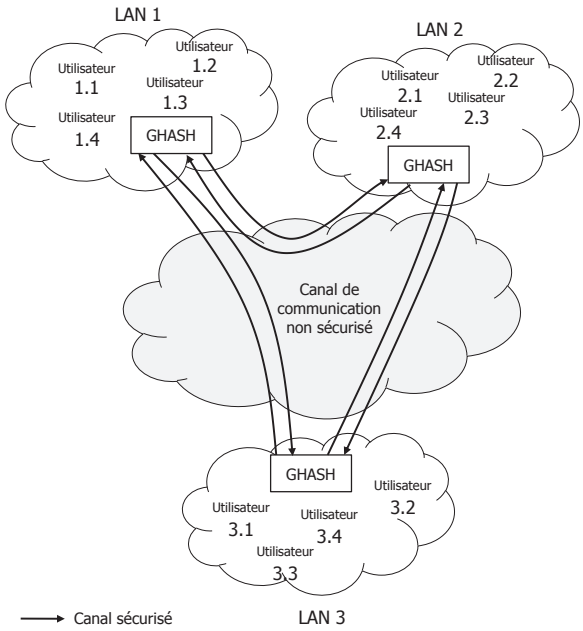

FIGURE 3.7 – Aperçu de la connectivité typique d'un réseau VPN étant authentifié avec GHASH

des débits de données authentifiées 2× supérieures aux approches publiées précédemment avec une surface réduite de 50 %. Les expériences avec un dispositif Spartan-3 montrent un débit significatif (14 Gbit/s) avec faible impact en surface (4875 LUTs).

GHASH sur ASIC : L'idée même de considérer une implémentation de GHASH sur les technologies ASIC est intéressante. Kuon et Bose (Kuon *et al.*, 2006) ont estimé en 2006 qu'un circuit sur FPGA est quarante fois plus large, trois fois plus lent et consomme douze fois plus de puissance dynamique comparé à un circuit similaire sur ASIC. Il est cependant important de relever que le changement de clé, revient à un changement de circuit, et le coût des masques associés est évidemment prohibitif. Toutefois, la fonction GHASH peut également être utilisée pour effectuer du hachage non cryptographique de qualité. Son application est donc variée et ne se cantonne pas au simple domaine qu'est la cryptographie".

Chapitre 4

Sécurité Configurable dans les Systèmes Embarqués

Des tendances actuelles, se dessine la volonté de voir les microprocesseurs remplacer les transistors. Certes, non pas en tant que composant atomique de fabrication mais comme unité macroscopique de contrôle et de calcul, par leur disponibilité en grande quantité, leur coût réduit et leurs fonctions spécifiques. L'approche décrite dans ce chapitre, se concentre spécifiquement sur la sécurité de ces dispositifs et particulièrement sur deux aspects : 1) la protection du chargement et 2) l'exécution sécurisée d'application. Basés sur des politiques de sécurité configurables, les bénéfices de cette protection faible pénalité sont démontrés par une implémentation sur plate-forme de prototypage.

Ce chapitre est constitué de neuf parties. La première propose une introduction à la sécurité des microprocesseurs. La seconde situe le contexte de ces travaux qui sont comparés par un état de l'art présent dans la troisième partie. Nos propositions sont décrites et étudiées dans les parties quatre et cinq, et sont suivies de trois parties, six, sept et huit, concernant la concrétisation expérimentale et les résultats obtenus. Le chapitre est finalisé par une conclusion.

Sommaire

4.1 Sécurité des microprocesseurs

Les systèmes embarqués sont largement utilisés pour exécuter des applications utilisateur dans une large gamme d'environnement allant des systèmes portables au contrôle automobile. Ces systèmes contiennent souvent un microprocesseur, de la logique configurable, de la mémoire externe, des ports d'Entrées/Sorties (E/S, Input/Output, IO) et des interfaces capteurs. En plus, des préoccupations standards concernant les performances et la consommation, la sécurité est devenue l'un des problèmes majeurs des applications embarquées. La nature portable de beaucoup de systèmes embarqués les rend particulièrement vulnérables aux attaques matérielles et logicielles. Dans beaucoup de cas, les adversaires ne sont pas intéressés par les détails de la conception, mais plus par le désir d'obtenir des informations sensibles du code programme ou des données qui sont présentes en mémoire. S'ils ne sont pas protégés, les transferts vers la mémoire hors-puce à partir du microprocesseur peuvent être facilement observés et peut révéler des informations importantes. Quelques protections mémoires peuvent être faites par simple chiffrement des données avant tout transfert vers la mémoire externe.

Le contenu des mémoires des systèmes embarqués requiert bien souvent d'être protégé tout au long des différentes phases, du chargement d'application jusqu'à l'état stable du système. Pour les microprocesseurs, le code de l'application est le plus souvent gardé dans une mémoire flash sur-carte pour faciliter le chargement. Après l'initialisation, ce code et les données sont stockés dans une mémoire externe principale qui est interfacée au microprocesseur via un bus vulnérable. La sensibilité des informations stockées varie de hautement sensible à peu importante, ce qui motive une protection de la mémoire configurable qui peut être ajustée par tâche ou par application. Les performances en lecture de beaucoup de systèmes embarqués indiquent la nécessité de mécanismes de sécurité directement implémentés sur-puce, étant adjacent au microprocesseur. Cette implémentation matérielle doit satisfaire les besoins en sécurité d'une application tout en minimisant la surface demandée. L'efficacité d'utilisation des ressources est typiquement l'une des clés pour le calcul des systèmes embarqués dans un environnement contraint.

Ces travaux inclus le développement d'une nouvelle approche de sécurité légère pour les systèmes embarqués qui contiennent des microprocesseurs implémentés dans des FPGAs. Notre approche offre de la sécurité pour les

accès aux instructions et données situées hors-puce pendant les phases de chargement d'applications et d'exécutions. Notre technique est différente des précédentes propositions pour la sécurité des mémoires (Elbaz *et al.*, 2006), (Lie *et al.*, 2003) (Suh *et al.*, 2005), parce qu'elle limite la pénalité d'utilisation des ressources logiques et stocke les tags de sécurité sur-puce. Après le chargement du code de l'application de la mémoire flash, le coeur de sécurité basé sur le mode AES-GCM (Nat, 2007) détermine le niveau de sécurité approprié. Pour faciliter le chargement de l'application durant le boot, un circuit léger de déchiffrement et d'authentification a été implémenté et il peut être réutilisé pour la protection mémoire de manière complètement transparente.

L'efficacité de cette approche est prouvée par l'évaluation des pénalités matérielles nécessaires par le système de sécurité complet pour 4 applications multi-tâches avec niveau de sécurité différent. Ces 4 applications ont été quantifiées et testées avec un processeur soft-core Microblaze implémenté sur un FPGA Xilinx issu de la famille Spartan-6. Le Microblaze exécute le système d'exploitation (Operating System, OS) MicroC/OS II (LaBrosse, 2002) pour l'ordonnancement des tâches. Une réduction moyenne des performances d'exécution de l'ordre de 13% est alors noté. Ces travaux fournissent les contributions suivantes :

– Notre approche fournit un chiffrement-authentifié faible-coût pour les systèmes embarqués basés sur la technologie FPGA. Le mode utilisé, AES-GCM, est approuvé par la NIST. L'approche minimise la logique et la latence requise pour l'authentification et s'appuie sur l'augmentation de capacité de mémoire interne au FPGA pour stocker les informations d'authentification.

– A l'instar des autres approches qui ciblent les systèmes embarqués (Vaslin *et al.*, 2008), le chiffrement-authentifié basé sur AES-GCM est synchronisé et parallélisé pour améliorer le débit.

– Les pénalités et performances ont été complètement implémentées et évaluées en matérielles pour, à la fois l'aspect chargement sécurisé d'application et l'aspect mémoire sécurisée.

– L'approche est flexible et permet la sélection du chiffrement et de l'au-

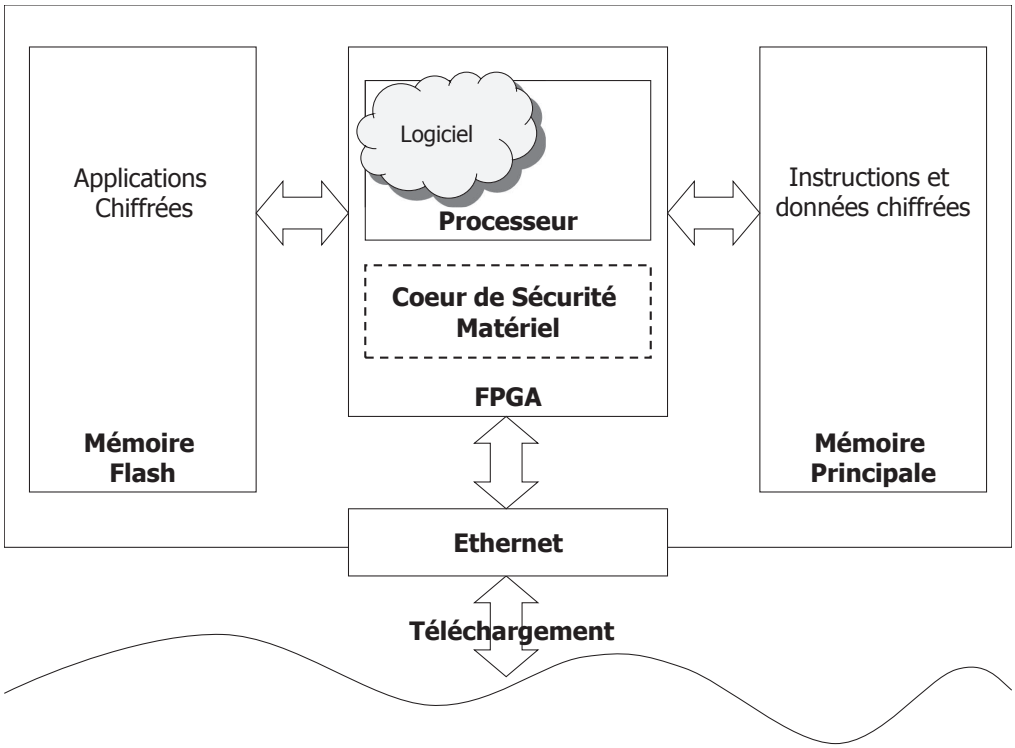

FIGURE 4.1 – Modèle haut niveau d'un système embarqué typique

thentification sur différentes parties de l'application. Ceci se fait sans addition d'instruction processeur ou d'ajout significatif pour le système d'exploitation.

L'approche peut être utilisée pour protéger un FPGA quelconque basé sur la technologie SRAM qui supporte le chiffrement du bitstream. Cette capacité est disponible pour les familles Altera Stratix II, III, et V et les familles Xilinx Virtex-2, -4, -5 et 6.

4.2 Contexte

4.2.1 Modèle du système

Le modèle du système qui est adressé par ce travail est montré en Figure 4.1. Le système embarqué est connecté avec l'extérieur via un port de communication (e.g. une communication Ethernet ou sans fil 802.11, etc...). Un microprocesseur sur puce exécute une ou plusieurs applications qui sont stockées dans une mémoire hors-puce. Après la mise en service ou la mise à zéro système, le code de l'application protégé est chargé de la mémoire flash vers la mémoire système, dans notre cas de type dynamique à débit de données double (Double Data Rate Synchronous Dynamic Random Access Memory, DDR-SDRAM), par le microprocesseur implémenté dans le FPGA. Ce modèle a été utilisé par une série d'études précédentes sur les systèmes embarqués (Lee et Orailoglu, 2008), (Pasotti *et al.*, 2003). Généralement, l'exécution du code n'est pas effectuée de la mémoire flash car elle introduit de fortes latences de lecture. Le temps nécessaire pour charger l'application est dépendant de la taille du code et de la latence de fetch des données. Pendant l'exécution, les instructions et les données sont stockées en DDR-SDRAM. Ces accès hors-puce sont fait de lectures et d'écritures.

L'aspect mémoire sécurisé décrit ici adresse deux scénarii de protection mémoire :

– Chargement système d'une application du microprocesseur : Pour fournir la sécurité du code de l'application, les contenus de la mémoire flash doivent être protégés contre les attaques. Puisque le code est stocké en mémoire flash, la même séquence est effectuée à la mise sous tension

	PE-ICE (Elbaz *et al.*, 2006)	XOM (Lie *et al.*, 2003)	AEGIS (Suh *et al.*, 2003b)	Yan-GCM (Yan *et al.*, 2006)
Sécurité	$\frac{1}{2^{32}}$	$\frac{1}{2^{128}}$ or $\frac{1}{2^{160}}$	$\frac{1}{2^{160}}$	$\frac{1}{2^{128}}$

Tableau 4.1 – Sécurité des systèmes contre des attaques par force brute

du système.

– Protection système de la mémoire principale : Après le chargement de l'application, le code et les données stockées en DDR-SDRAM doivent être protégés.

Ces mécanismes de sécurité doivent consommer un minimum de surface et de puissance et produire des performances en adéquation avec les systèmes embarqués. Bien que la Figure 4.1 montre la capacité potentielle de télécharger de manière sécurisée de nouvelles applications d'une source externe à la mémoire flash ou DDR-SDRAM, ce problème n'est pas adressé par ce travail.

4.2.2 Menaces sur les mémoires des systèmes embarqués

La mémoire principale d'un système embarqué peut rencontrer un nombre varié d'attaques (Elbaz *et al.*, 2006) qui résultent soit de l'écoute d'une interface entre un processeur et la mémoire, soit d'une attaque physique (injection de fautes). L'écoute de bus permet la collection de valeurs d'adresses et de données pouvant être utilisées pour révéler le comportement du processeur. Pour garantir la confidentialité, le chiffrement des données se fait par des algorithmes tels que AES ou 3DES avant les transferts extérieurs. Les données chiffrées par ces algorithmes ne peuvent être déchiffrées sans la clé associée. Cependant, même les données chiffrées et leurs adresses associées peuvent être vulnérables aux attaques. Une attaque par falsification apparaît lorsqu'un adversaire place une valeur de donnée aléatoire en mémoire, ce qui provoque un mauvais fonctionnement du système. Une attaque en relocation, apparaît lorsqu'une donnée valide est copiée en une ou plusieurs locations mémoires. Une attaque en rejeu apparaît lorsqu'une donnée, qui a été précédemment stockée en mémoire, substitue la donnée courante. Les instructions du processeur sont particulièrement vulnérables aux attaques en relocation parce que les séquences d'instructions spécifiques peuvent être répétées pour forcer le système dans un état particulier. Des approches spécifiques pour maintenir l'authentification des données sont nécessaires contre ce type d'attaque. L'authentification dans ce contexte indique que les données récupérées d'une position mémoire sont les mêmes que les données les plus récemment écrites.

4.2.3 Modèle de menace du système

La portée de notre travail sur la sécurité de la mémoire principale est limitée au même modèle de menace que les approches précédentes (Elbaz *et al.*, 2006) (Suh *et al.*, 2003b) (Lie *et al.*, 2003). Concernant le modèle de menace, les hypothèses spécifiques sont faites :

- Le FPGA et son contenu (e.g. un microprocesseur) sont sécurisés, et les clés ainsi que les informations de configurations et informations utilisateur du FPGA, ne sont pas accessibles par attaque physique ou logique. Ces attaques inclues les attaques par puissance différentielle (DPA), attaque par canaux cachés et l'écoute. Le FPGA est une zone de confiance.

- Le bitstream de configuration est chiffré et stocké extérieurement au FPGA (e.g. dans une mémoire flash sur-carte). Le bitstream est déchiffré dans le FPGA par les techniques de déchiffrement rendues disponibles par les entreprises fournisseurs de FPGAs. Un nombre effectif de techniques de chiffrement de bitstream (Xil, 2005) et de téléchargement de bitstream (Badrignans *et al.*, 2008) ont été développés et testés pour des dispositifs commerciaux.

- Les composants extérieurs au FPGA ne sont pas de confiance. Ces ressources incluent la mémoire DDR-SDRAM, la mémoire flash qui contient le code de l'application et la mémoire flash qui contient les informations du bitstream. Ces composants peuvent être sujet à des attaques physiques dans le but de lire et/ou de modifier les données qui y sont présentes.

- Les interconnexions entre les composants et le FPGA sont aussi vulnérables aux attaques. Les données circulant sur les interconnexions peuvent être observées et modifiées par un adversaire.

Les interconnexions et les composants en-dehors du FPGA sont logés dans une zone n'étant pas de confiance. Notre approche apporte la protection contre les attaques par falsification, par relocation et par rejeu.

4.2.4 Travaux précédents

4.2.4.1 Sécurité de la mémoire principale

Des techniques ont été développées pour apporter la confidentialité des données et l'authentification à la mémoire principale dans les systèmes basés sur des processeurs. Pour ces systèmes (Elbaz *et al.*, 2006) (Lie *et al.*, 2003) (Suh *et al.*, 2005) (Suh *et al.*, 2003b) (Yan *et al.*, 2006), la confidentialité est donnée par le chiffrement des données par l'utilisation d'algorithme AES ou 3DES. Les données sont chiffrées avant tout transfert hors-puce et déchiffrement lors de la lecture de la mémoire. L'authentification de données est typiquement maintenue par le hachage de ces valeurs de données et est faite de manière hiérarchique. Le niveau de sécurité force brute de ces schémas, résumé dans le Tableau 4.1, mesure la probabilité que pourrait avoir un adversaire à casser l'authentification en changeant une valeur de donnée mais qui passerait la vérification de l'authentification (Authentication Check, AC). Ce type d'attaque requiert un grand nombre de tentative avec des valeurs de données variées. Même si les solutions précédentes ont été montrées comme efficaces, le coût de la sécurité peut être élevé en terme de mémoire nécessaire pour stocker les hachés (Gassend *et al.*, 2003), les hachés compressés (Suh *et al.*, 2003b), et les tags AC pour chaque donnée et augmente la latence de lecture à cause des AC. Notre nouvelle approche limite le temps d'authentification bien qu'elle requiert le stockage des informations de sécurité, sur-puce. Cette pénalité est quantifiée dans la Section 4.6. L'utilisation de l'AES-GCM pour effectuer l'authentification accélérée des accès en mémoire externe a été proposée premièrement par (Yan *et al.*, 2006). Ce travail implique la simulation d'un système basé sur un microprocesseur, et inclu une hiérarchie de cache multi-niveau. Notre travail se concentre sur la quantification des performances et du coût en surface d'une sécurité basée sur AES-GCM quand les ressources de calcul sont limitées, typiquement dans un système embarqué.

4.2.4.2 Chargement sécurisé d'application

Le chargement d'application à partir d'une mémoire flash pour un système basé sur un microprocesseur a été largement examiné dans le contexte des ordinateurs personnels (Arbaugh *et al.*, 1997) et des dispositifs mobiles (Dietrich et Winter, 2008). Comme notre approche, les techniques de (Heath et Klimov, 2006) incluent le code dans une mémoire flash qui est externe au processeur. Ce code est chiffré et protégé par des valeurs supplémentaires

d'authentification stockées également en mémoire flash. Les premières approches de chargement sécurisé (Arbaugh *et al.*, 1997) intégraient l'authentification dans le système d'entrée/sorties du processeur (Basic Input Output System, BIOS) pour assurer le chargement correct. Ce processeur est généralement pensé comme étant complexe pour les systèmes embarqués (Dietrich et Winter, 2008). Les approches les plus récentes, comme TrustZone, (Alves et Felton, 2004), dépassent la nécessité d'une mémoire d'authentification supplémentaire en lecture seule (Read-Only Memory, ROM) en intégrant cette ROM sur la même puce que le processeur. Bien qu'efficaces, les systèmes embarqués ne contiennent pas tous des FPGA contenant de la mémoire ROM. Une réévaluation récente du chargement sécurisé d'application pour les plateformes mobiles (Dietrich et Winter, 2008) note que la flexibilité logicielle et matérielle est possible dans l'implémentation d'un système de chargement sécurisé dans les systèmes embarqués. Le groupe calcul de confiance (Trusted Computing Group, TCG) définit une hiérarchie des activités de chargement qui inclue la récupération initiale du code du processeur. Notre implémentation se concentre sur le plus bas niveau de la hiérarchie, la lecture du code pour un processeur situé dans un endroit externe et non sécurisé. A l'inverse des autres techniques, notre approche cible spécifiquement la minimisation de la logique supplémentaire nécessaire pour le processus de chargement sécurisé.

4.3 Précisions sur les solutions académiques et industrielles

La majeure partie de la recherche sur les processeurs sécurisés s'est concentrée sur un seul microprocesseur. Ce type de système englobe beaucoup de systèmes à base de processeurs généralistes et de systèmes embarqués. La sécurité peut être vu du côté matériel et du côté logiciel. Pour ce dernier, la sécurité est approchée de manière statique ou de manière dynamique (Milenkovic, 2005). Les approches matérielles sont fondées principalement sur l'ajout de silicium dédié avec un degré plus ou moins fort de support logiciel. Cette section se concentre sur les approches matérielles puisque notre architecture est orientée vers cette spécificité.

Bien des techniques se proposent d'adresser les types communs d'at-

taques. (Xu *et al.*, 2002) et (Ozdoganoglu *et al.*, 2006) proposent de se prémunir des attaques de type débordement de pile par l'utilisation d'une pile matérielle. (Tuck *et al.*, 2004) suggère d'utiliser des pointeurs d'adresses chiffrés. (Suh *et al.*, 2004) et (Crandall et Chong, 2004) proposent de tagger les données transitant sur un canal n'étant pas de confiance et d'interdire leur utilisation en tant que cible de saut. (Barrantes *et al.*, 2003) rendent aléatoire le jeu d'instruction d'un processeur pour rendre les attaques plus difficiles. Quelques techniques adressent les attaques par canaux cachés comme dans (Wang et Lee, 2007) qui partitionne les caches pour prévenir les analyses de défaut de cache et (Ambrose *et al.*, 2007) injecte du code aléatoire pour éviter les attaques par analyse de consommation.

Notre approche a pour but d'être plus complète que les propositions cidessus. Pour cela nous allons examiner de manière plus précise les approches qui sont de cet acabit dans le domaine monoprocesseur, l'aspect multiprocesseur étant étudié dans l'Annexe. Les solutions apportées pour les architectures monoprocesseurs seront divisées en deux sous-parties. La partie académique est bien documentée, c'est moins le cas pour la partie industrielle par sa nature souvent propriétaire. Nous examinerons également les solutions qui ciblent la logique reconfigurable. Le lecteur pourra également se référer à la thèse de (Rogers, 2010) pour de plus amples détails.

4.3.1 Propositions liées à l'académique

(Ragel et Parameswaran, 2006) introduit une architecture pour vérifier l'intégrité du code. Le compilateur calcule la somme de contrôle pour chaque bloc de base. Des instructions spéciales sont insérées au début de chaque bloc pour charger la somme de contrôle dans un registre dédié. La somme est indépendamment calculée lors de l'exécution du bloc en question et lorsqu'une instruction qui altère le flot de contrôle est rencontrée, la somme de contrôle calculée est comparée avec la somme précédemment chargée. Si elles ne correspondent pas, le bloc a subit une altération. Cette approche requiert un support logiciel et matériel. Ceci inclut des modifications du compilateur et l'ajout d'instructions personnalisées. De plus, ceci ne cible que l'intégrité des instructions, aucune forme de confidentialité et d'intégrité des données n'est proposée.

Le modèle exécution en mémoire seulement (eXecute Only Memory, XOM)

proposé dans (Lie *et al.*, 2003) décrit une architecture qui convient aux besoins que sont la confidentialité et l'intégrité. La mémoire principale est considérée comme étant non sécurisée. Toutes les données qui entrent ou sortent du processeur sont alors chiffrées. Dans sa forme originale, l'architecture était vulnérable aux attaques par rejeu. Cette vulnérabilité fut comblée dans (Lie *et al.*, 2000). Les inconvénients de cette proposition sont sa complexité et la pénalité en performances. XOM requiert une modification du coeur du processeur, de ses caches associés et une addition de matériel dédié sécurité. L'architecture produit une pénalité en performance de l'ordre de 50%, valeur estimée par les concepteurs eux mêmes.

La forte pénalité introduite par XOM est réduite par les améliorations architecturales proposées par (Yang *et al.*, 2005). Ils adressent uniquement la confidentialité puisqu'ils se basent sur XOM qui adresse déjà les problèmes liés à l'intégrité. Ils proposent d'utiliser le schéma masque jetable (One-Time Pad, OTP) pour le chiffrement et le déchiffrement. Seul le masque est chiffré et l'on effectue un ou-exclusif entre celui-ci et le texte en clair pour produire le texte chiffré, ou avec le texte chiffré, pour obtenir le texte en clair. Ils augmentent la sécurité des données en incluant un nombre dit de séquence dans le masque pour les blocs de données et ajoutent une mémoire sur puce pour cacher ces nombres. Bien que leur schéma améliore largement les performances de XOM, il hérite également de ses autres faiblesses.

(Gassend *et al.*, 2003) propose de vérifier la mémoire qui n'est pas de confiance en utilisant un arbre de hachés. Il adresse uniquement l'intégrité en pointant que leur architecture peut être ajoutée à un système comme XOM qui peut gérer les préoccupations liées à la confidentialité. L'utilisation d'un arbre de hachage introduit une pénalité significative sur la bande passante qui peut être partiellement limitée par l'intégration du mécanisme de hachage au sein même des caches systèmes. Cependant, la pénalité de leur proposition est tout de même élevée avec un maximum de 20% pour l'architecture la plus efficace.

(Lu *et al.*, 2006) propose une architecture similaire qui utilise les codes d'authentification de messages en arbre. Ils sont calculés pour chaque bloc de cache qui incorpore son adresse virtuelle et une clé secrète d'application. L'architecture en arbre ne montre pas d'amélioration par rapport à l'approche dans (Gassend *et al.*, 2003) qui introduit pourtant une pénalité en perfor-

mance moyenne de 10% à 20%.

(Platte et Naroska, 2005) décrit un autre système de signature-et-vérification basé sur les arbres pour protéger l'intégrité du code et des données, mais également les valeurs des registres durant une trappe du système d'exploitation. Il traite dynamiquement le code généré de la même manière que les données dynamiques mais n'autorise pas l'utilisation de bibliothèques liées dynamiquement. Leur conception adresse seulement l'intégrité et n'assure pas la confidentialité. De plus, la vérification n'est pas immédiate. La vérification des blocs de données est seulement garantie comme complète qu'après la séquence d'appel ou le changement de contexte suivant. Ceci ouvre une fenêtre de vulnérabilité durant laquelle des instructions malicieuses peuvent être exécutées sans vérification. Parce que les registres sont sécurisés, le compilateur et le système d'exploitation doivent être modifiés pour utiliser les instructions ajoutées pour accéder aux données sécurisées. Aucune analyse de performance n'est présentée.

(Elbaz *et al.*, 2006) développe une technique pour effectuer le déchiffrement et l'intégrité de manière parallèle. Il s'appuie sur l'avantage de la propriété de diffusion de l'algorithme AES où chaque bit d'un bloc de texte clair influence tous les bits du bloc du chiffré correspondant. Avant chaque chiffrement, un nonce aléatoire est ajouté à tous les blocs protégés de données. Les nonces sont stockés sur-puce et lorsqu'un bloc protégé est déchiffré, le nonce du texte (message) clair est comparé avec le nonce précédemment stocké. Si les nonces correspondent, alors le bloc est valide pour utilisation. Par simulation, les auteurs notent une pénalité moyenne de 4%. Cette approche nécessite une méthode de génération de nonces tout comme des ressources sur-puce pour stocker la table des nonces attendus. Cependant, cette proposition élimine la nécessité d'une structure type arbre en mémoire. Cette architecture est étendue dans (Elbaz *et al.*, 2007) pour supporter le stockage de nonces hors-puce. Dans ce cas, le nonce est formé d'un bloc d'adresse protégé et d'une valeur de compteur. Une structure en arbre est utilisée pour protéger les valeurs de compteur. Leur approche introduit une pénalité en mémoire de 100% et aucune évaluation de performance n'est proposée.

(Suh *et al.*, 2005) propose une architecture pour adresser la confidentialité et l'intégrité. Leur architecture utilise le chiffrement par masques jetables pour fournir l'aspect confidentialité avec une pénalité relativement faible. Ce-

pendant, comme leurs fonctions cryptographiques prennent un horodatage comme entrée, ils proposent que la mémoire protégée soit, en intégralité, rechiffrée lorsqu'un compteur effectue un dépassement de capacité. Pour réduire cette pénalité introduite lors de la vérification d'intégrité, ils proposent la construction d'un journal des accès en mémoire en utilisant des fonctions de hachage incrémentales multi-ensemble décrites dans (Suh *et al.*, 2003a). Ils admettent qu'un programme produit des sorties significatives et signées à la fin de son exécution ou à des intervalles de temps discrets. Leur architecture vérifie les séquences d'accès à la mémoire hachée uniquement lorsque ces sorties sont produites. La vérification étant peu fréquente, la pénalité est négligeable. L'inconvénient majeur est que la falsification n'est pas immédiatement évidente ce qui rend le système potentiellement vulnérable pendant les vérifications.

Le travail de (Milenković *et al.*, 2005b) propose une architecture qui adresse uniquement l'intégrité des instructions et implique la signature des blocs instructions durant une procédure d'installation sécurisée. Ces signatures sont calculées en utilisant les mots de l'instruction, l'adresse de début du bloc et une clé secrète associée au processeur. A l'exécution, ces signatures sont recalculées et vérifiées avec les signatures récupérées en mémoire. La fonction cryptographique utilisée au sein de l'architecture est une simple fonction polynomiale implémentée à l'aide de multiples registres à décalages. Cette architecture est mise à jour dans (Milenković *et al.*, 2005a) et (Milenkovic *et al.*, 2006), et ajoute un chiffrement AES pour renforcer la sécurité et embarque les signatures avec les blocs d'instruction au lieu de les stocker dans une table. Parce qu'une seule clé est utilisée par tous les programmes, cette architecture reste vulnérable aux attaques en relocation (splicing).

(Drinic et Kirovski, 2004) proposent une architecture similaire mais avec une utilisation du chiffrement basée sur le mode CBC-MAC et ils incluent les signatures dans les lignes de cache. Il propose de réduire la pénalité en performances, en ordonnant les blocs de manière à ce que les instructions qui ne seraient pas sûre dans le cas d'une exécution sécurisée, ne soient pas émises tant que la verification de la signature n'est pas complète. Le désavantage de cette approche est la nécessité d'un support compilateur important et ne peut masquer la pénalité de vérification. De plus, leur architecture n'adresse pas la confidentialité et est vulnérable aux attaques de rejeu et en relocation.

La recherche jointe entre l'Institut Technologique de Géorgie (Georgia Institute of Technology, GA Tech) et l'Université de Caroline du Nord (North Carolina State University, NCSU) a proposé plusieurs conceptions. (Yan *et al.*, 2006) décrit une architecture signature-et-vérification qui utilise le mode GCM. Ils protègent les données en divisant les nombres de séquence pour réduire la pénalité en mémoire et redire la probabilité qu'une séquence déborde de sa plage de valeurs. Une structure en arbre est utilisée pour protéger les valeurs dynamiques contre les attaques de rejeu. (Rogers *et al.*, 2007) réduit la pénalité de cette architecture en restreignant l'arbre pour ne protéger que les nombres de séquence. Ils annoncent une pénalité en performances de 11.9%. Cette pénalité peut être artificiellement basse parce qu'ils utilisent un système de vérification de l'intégrité qui est peu précis. Ceci autorise l'exécution d'instructions potentiellement dangereuses avant vérification complète.

4.3.2 Propositions liées à l'industrie

Les fondeurs Intel et Advanced Micro Devices (AMD) ont tous deux introduits des caractéristiques pour notamment prévenir des attaques par dépassement de tampon. Intel appelle cette capacité Désactivation du Bit d'Exécution (Intel), qui interdit au processeur d'exécuter des instructions qui ont pour origine certaines parties de la mémoire. AMD propose une approche similaire (Zeichick) où un bit est stocké dans la table des pages et, est vérifié lorsqu'un défaut a lieu sur le tampon de traduction des adresses virtuelles en adresses physiques (Translation Lookaside Buffer, TLB). Intel et AMD autorisent la désactivation de cette fonctionnalité de façon logicielle.

International Business Machines (IBM) a développé l'architecture SecureBlue (IBM). Comme les propositions académiques précédemment décrites, elle se base sur la cryptographie pour assurer l'intégrité et la confidentialité du code et des données. SecureBlue est destinée à être incorporée aux microprocesseurs existants.

Advanced RISC Machines (ARM) commercialise l'architecture sécurisée TrustZone (Alves et Felton, 2004), prévue pour "augmenter" les microprocesseurs ARM. Elle repose sur un support logiciel et matériel. Le composant matériel utilise la cryptographie pour adresser l'intégrité et la confidentialité, et compartimente l'exécution du microprocesseur en deux modes, sécurisés

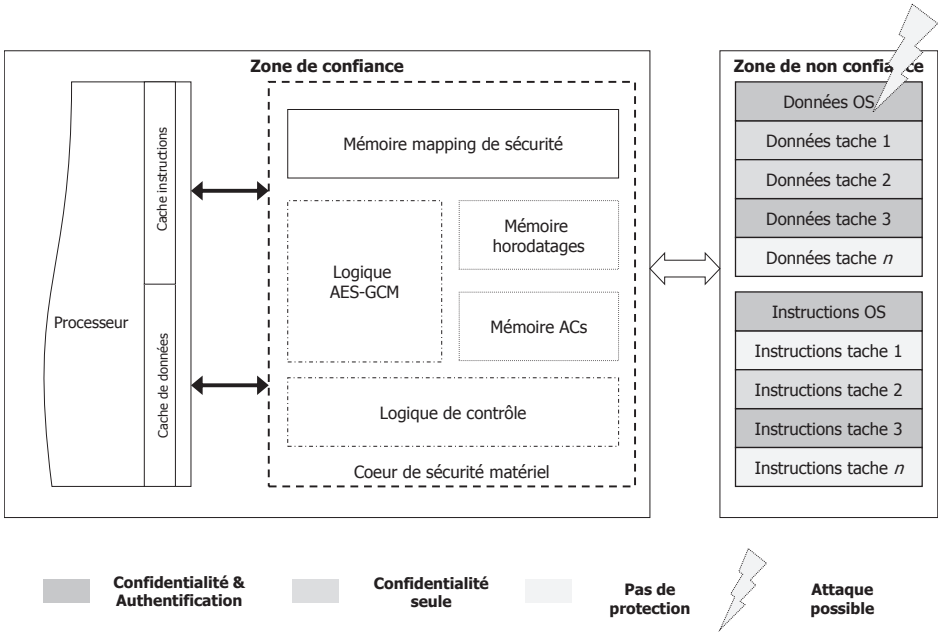

FIGURE 4.2 – Aperçu de la protection de la mémoire principale

ou non. La partie logicielle inclue un moteur TrustZone qui s'ajoute au système d'exploitation et fournit une interface de programmation (Application Programming Interface, API) pour l'écriture de programmes sécurisés.

Maxim (anciennement Dallas Semiconductor) fabrique le microprocesseur sécurisé DS5250 (MAXIM). Il est conçu pour servir en tant que co-processeur pour les systèmes embarqués possédant des microprocesseurs non sécurisés traditionnels. Maxim propose que le co-processeur exécute des fonctions à caractère sensible pendant que le processeur principal effectue les opérations les moins critiques. Le DS5250 contient de la mémoire sur-puce non volatile qui est effacée si une falsification physique est détectée. Cette mémoire est utilisée pour stocker la clé secrète du processeur, et peut également être utilisée pour stocker d'autres données sensibles. Le DS5250 peut également accéder à de la mémoire externe de manière sécurisée.

Secure Machines propose une architecture pour sécuriser les systèmes embarqués comme ceux contenus dans les téléphones portables (Perrotey et Bressy). Leur architecture cible des systèmes complets et assure la communication sécurisée entre les différents périphériques. Cependant, cette sécurité demande que toutes les puces utilisées dans le système partagent un jeu de clés commun et contiennent des machines d'états de sécurité appelées contrôleurs sécurisés matériels. Un noyau sécurisé tournant sur le microprocesseur interagit avec les contrôleurs sécurisés mais aucun détail n'est spécifié.

4.3.3 Propositions ciblant la logique reconfigurable

Quelques chercheurs ont ciblé le domaine de la logique reconfigurable. (Su *et al.*, 2005) ont développé un co-processeur cryptographique sur un FPGA pour accélérer ces fonctions dans un système embarqué. (Zambreno *et al.*, 2005) propose l'utilisation des FPGA comme un intermédiaire qui analyse toutes les données récupérées par un processeur. Ils calculent des sommes de contrôle en utilisant deux méthodes comme le hachage du code et de la liste des registres utilisés par les instructions et comparent les deux sommes. Le niveau de sécurité fournit par cette approche est une question ouverte et un support compilateur étendu est nécessaire. On notera notamment l'insertion d'instructions factices. Ceci mène à une pénalité assez élevée de 20% et ne permet que l'intégrité et la confidentialité des instructions.

(Suh *et al.*, 2005) développe le processeur sécurisé AEGIS sur un FPGA. Il décrit les fonctions physiquement non duplicables (Physically Unclonable Functions, PUF) pour générer les secrets nécessaires. La mémoire est divisée en quatre régions basées sur l'aspect statique ou dynamique (lecture-seule ou écriture-seule) et sur l'utilisation de l'intégrité seulement ou de l'intégrité et de la confidentialité. Ils autorisent les programmes à changer de mode de sécurité à l'exécution en commençant dans un mode standard non sécurisé et en passant entre un mode supportant de l'intégrité seulement à un mode supportant l'intégrité et la confidentialité. Ils autorisent également la suspension des modes sécurisés par des appels à des bibliothèques. Cette flexibilité a un coût, et leur architecture admet un support important du système d'exploitation et du compilateur.

4.3.4 Relation avec le travail précédent des auteurs

Ce travail étend les travaux de (Vaslin *et al.*, 2008) sur la sécurité de la mémoire principale décrit en intégrant la protection de la mémoire et le chargement sécurisé d'application. Ces deux composants de sécurité sont complémentaires l'un de l'autre et permettent le partage de ressources. L'authentification de la mémoire système est effectuée par le mode AES-GCM, un bloc de chiffrement qui est prouvé sécurisé (Nat, 2007). Les travaux précédents utilisaient une approche ne permettant pas d'assurer une sécurité suffisante puisque basée sur des opérations AES abrégées.

4.4 Architecture de la sécurité pour la mémoire principale

4.4.1 Politique de sécurité

Avant de discuter notre approche sur le chargement d'application sécurisée à partir de la mémoire flash, nous décrivons la sécurité de la mémoire centrale. Notre approche est montrée en Figure 4.2. et se base sur un coeur de sécurité matériel (Hardware Security Core, HSC) façonné à partir de logique et de mémoire embarquées qui est capable de gérer différents niveaux de sécurité dépendant de l'adresse mémoire reçue par le processeur.

Un segment mémoire, qui correspond à une tâche logicielle définie par l'utilisateur, peut être déterminé à partir du code compilé ou de données associées. Selon la politique de sécurité requise par le concepteur logiciel, un nombre important de segments mémoires peut être construit. Chacun de ces segments est définit par 4 paramètres :

- L'adresse de base du segment.
- La taille du segment.
- Le niveau de sécurité du segment.
- Le type de segment : code (lecture seule) ou donnée (lecture et écriture).

Une table de correspondance est utilisée pour stocker et mapper ces segments en mémoire et est appelée mémoire de sécurité (Security Memory Map, SMM). Elle est inclue en matériel dans le coeur de sécurité matériel. Nous considérons trois niveaux de sécurité pour chaque segment mémoire : confidentialité uniquement, confidentialité et authentification, ou pas de sécurité. L'implémentation de la politique de sécurité dans la mémoire SMM est indépendante du processeur et du système d'exploitation associé. L'isolation de la mémoire SMM du processeur la rend sécurisée contre toute modification logicielle. Bien que l'authentification seulement peut être un autre niveau de sécurité possible, nous ne revendiquons pas le support de ce niveau car nous ne l'avons pas évalué expérimentalement. Puisque le coeur HSC fonctionne directement avec les adresses des segments mémoires, aucun compilateur spécifique pour l'étape de compilation est nécessaire.

4.4.2 Gestion des niveaux de sécurité

L'utilisation d'un système d'exploitation avec la plupart des processeurs des systèmes embarqués produit un partitionnement naturel du code de l'application et de ses données. Dans la figure 4.2, les instructions de l'application et les données de la tâche 1 ont des niveaux de sécurités différents et requièrent différents segments mémoires. La disponibilité de niveaux de sécurité configurables provoque un bénéfice important, si il n'est pas nécessaire d'effectuer systématiquement confidentialité et authentification. La quantité de mémoire sur-puce requise pour stocker les tags pour la vérification de l'au-

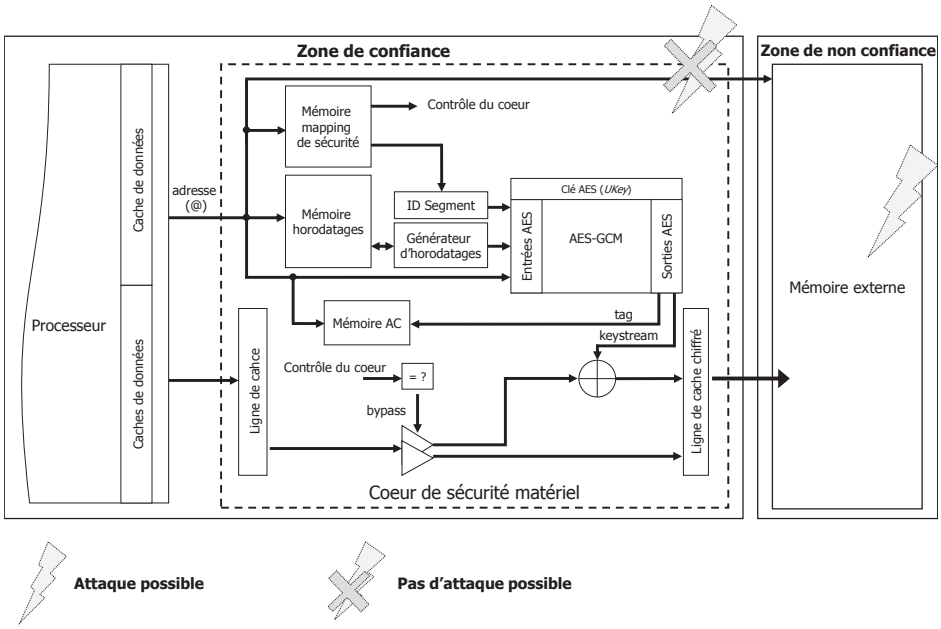

FIGURE 4.3 – Architecture du coeur de sécurité pour une écriture

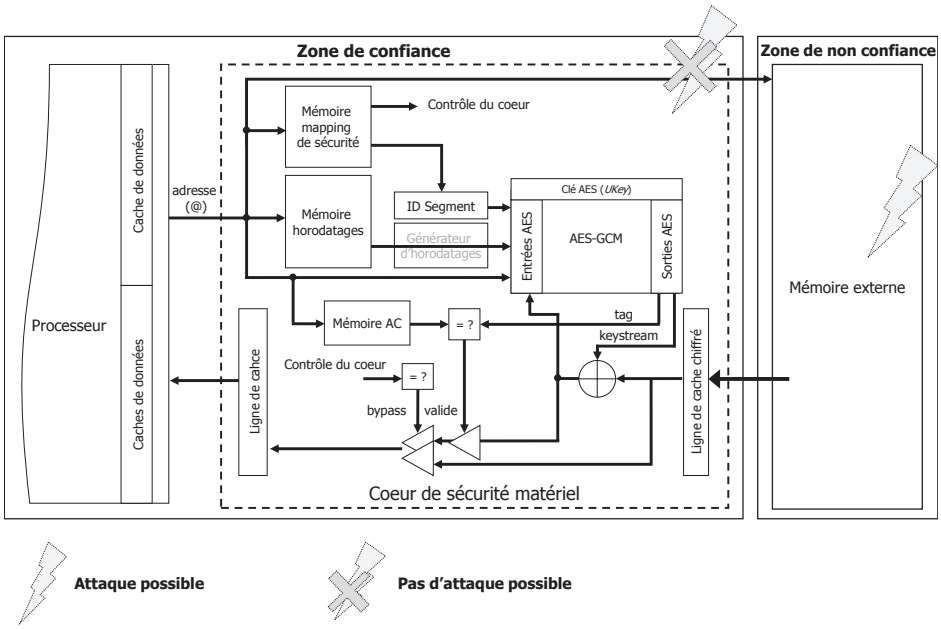

FIGURE 4.4 – Architecture du coeur de sécurité pour une lecture

thentification peut être réduite, si seules les parties sensibles de la mémoire externe doivent être protégées. De plus, la latence et la puissance dynamique des accès à la mémoire non protégée sont minimisées puisque les calculs en question sont évités.

4.4.3 Architecture du coeur de sécurité mémoire

Notre coeur pour la gestion des différents niveaux de sécurité est une extension des versions préliminaires (Vaslin *et al.*, 2008) qui offrent une sécurité uniforme pour toutes les tâches et segments mémoires, et utilisent les opérations de type OTP (Suh *et al.*, 2003b) pour la confidentialité et des séquences AES abrégées pour l'authentification. La confidentialité dans notre nouveau système emploie le standard NIST, (McGrew et Viega, 2004a), (Nat, 2007) qui est prouvé sécurisé. Contrairement aux autres modes de chiffrement comme ECB, CBC ou le mode compteur CTR, AES-GCM assure à la fois la confidentialité et l'authenticité (chiffré-authentifié) et peut être à la fois pipelinée et parallélisée.

Plutôt que de chiffrer les données d'écriture directement, notre approche génère un flux de clés avec l'algorithme AES qui opère avec une clé secrète stockée dans l'architecture. Dans notre implémentation, une valeur d'horodatage (TimeStamp, TS), l'adresse de la donnée, et l'identifiant ID du segment mémoire de la donnée, sont utilisés comme entrées d'un circuit de chiffrement AES-GCM pour générer le flux de clés. Un ou-exclusif est effectué entre ce flux de clés et la donnée pour générer le texte chiffré pouvant être transféré en dehors du FPGA (System-on-Chip, SoC) qui contient le microprocesseur. L'horodatage est incrémenté pour chaque écriture de ligne de cache. Le même ID de segment est utilisé pour toutes les lignes de cache appartenant à un segment particulier de l'application. Comme les implémentations précédentes basées sur les masques jetables (Suh *et al.*, 2003b), le bénéfice de cette politique d'exécution $Exec_{GCM}$ comparé à un chiffrement direct peut être vu pendant les phases de lecture de données. La génération du flux de clés peut commencer immédiatement après que l'adresse de lecture soit connue. Après la récupération de la donnée, une simple et rapide opération ou-exclusif est demandée pour retrouver le texte clair. Si un chiffrement direct était utilisé, le processus de déchiffrement demanderait bien plus de cycles d'horloge après l'arrivée jusqu'au processeur de la donnée. Une limitation de cette approche est le besoin de stocker des valeurs d'horodatages pour chaque donnée (usuel-

lement une ligne de cache) dans une mémoire sur-puce. Ceci pour valider la fraîcheur des données et contrecarrer les attaques par rejeu. Une vue haut niveau du placement de ces blocs de sécurité est donnée en Figures 4.3 et 4.4.

La Figure 4.5 montre les opérations AES-GCM nécessaires pour chiffrer une ligne de cache en clair de 256 bits (le même schéma est appliqué pour le déchiffrement). La figure montre deux opérations AES sur 128 bits E_{Ukey} utilisant toutes les deux une clé secrète de 128 bits, $UKey$. Une entrée 128 bits de AES inclue 32 bits pour la valeur TS, 32 bits pour l'adresse de la donnée (@) et 64 bits pour l'identifiant du segment mémoire (SegID). La valeur $0||Len(C)$ est un mot de 128 bits qui résulte du remplissage (padding) de la longueur du texte chiffré C avec des bits à zéro. Deux textes chiffrés de 128 bits et un tag d'authentification de 128 bits également sont générés à partir des deux valeurs d'entrées textes clairs de 128 bits. Le tag n'est pas chiffré puisqu'il est stocké dans une mémoire embarquée sur-puce qui est de confiance.

Une description étape par étape de la politique $Exec_{GCM}$ pour protéger les écritures et lectures de données avec le bloc AES-GCM de la Figure 4.5 est montrée par les Algorithmes 1 et 2. D'un point de vue sécurité, il est essentiel que le flux de clé appliqué pour le chiffrement des données ne soit utilisé qu'une seule fois. Comme le flux de clés est obtenu avec l'AES, les entrées de l'AES doivent être également utilisées qu'une seule fois. Si un même flux de clés est utilisé plusieurs fois, des fuites d'informations peuvent arriver et un adversaire peut déjouer à la fois le chiffrement et l'authentification. L'utilisation de l'adresse mémoire de la donnée dans le processus de génération des flux de clés (Figures 4.3 et 4.4) protège des attaques par relocation. Pour prévenir les attaques par rejeu, un simple compteur 32 bits ou un registre à décalage à rétroaction linéaire (Linear feedback shift register, LFSR) est choisi pour la génération de l'horodatage.

Comme montrée par l'Algorithme 1, la valeur TS de 32 bit est incrémentée de deux après chaque écriture en mémoire puisque deux valeurs TS sont utilisées par le circuit en Figure 4.5. Ces valeurs sont stockées dans une mémoire Timestamp et sont indexées par l'adresse mémoire des données. Pour chaque demande d'écriture d'une nouvelle ligne de cache, le système calcule un flux de clés différent puisque la valeur du TS est mise à jour. Durant une lecture, la valeur originale du TS est utilisée à des fins de comparaison (Algorithme

2). La valeur TS récupérée est fournie à l'AES pendant la requête de lecture. Cette valeur est rapportée de la mémoire d'horodatage en utilisant l'adresse mémoire de la donnée. Le résultat de l'opération AES donnera le même flux de clé que celui produit pour la requête en écriture et la donnée déchiffrée deviendra le texte clair après avoir subi l'opération ou-exclusif (étape 5 de l'Algorithme 2).

L'utilisation du coeur AES-GCM permet une authentification prouvée-sécurisée sans pénalité importante en latence. Si deux coeurs AES 128 bits sont utilisés pour le chiffrement et le déchiffrement de valeur de données de 256 bits (Figure 4.5), les processus de chiffrement authentifié et de déchiffrement authentifié peuvent être réalisés en 13 cycles : 10 cycles pour le chiffrement et le déchiffrement, et 3 cycles pour l'authentification. Chaque multiplication dans $GF(2^{128})$, lorsque implémenté avec un chemin de données complètement parallèle avec des opérations "ou-exclusif" et "et", prend 1 cycle (Nat, 2007). Les opérations ou-exclusifs visibles aux entrées de $Mult_H$ en Figure 4.5, sont implémentées directement dans le coeur $Mult_H$. Aucun cycle additionnel n'est nécessaire. Le nombre total de cycles décrit ici n'inclut pas la latence des transactions du bus.

Les données en lecture seule, comme les instructions processeurs, ne nécessitent pas de protection contre les attaques par rejeu parce que ces valeurs ne sont jamais modifiées. L'utilisation mémoire sur-puce est alors réduite en conséquence. Les données en lecture seule peuvent cependant être la cible d'attaques par relocation mais l'adresse utilisée par la politique $Exec_{GCM}$ garantit la protection contre ces attaques. Si une donnée est relogée, son adresse différera de celle utilisée pour générer le flux de clé. La valeur chiffrée sera alors invalidée.

Algorithme 1 - ***Requête en écriture :***

---

1 – *incrémentation horodatage* : $TS = TS + 2$
2 – $\{flux\ de\ clé,\ tag\} = AES-GCM\{SegID,\ @,\ TS\}$
3 – $texte\ chiffré = texte\ clair \oplus flux\ de\ clé$
4 – $texte\ chiffré \Rightarrow mémoire\ externe$
5 – *stockage horodatage* : $TS \Rightarrow mémoire\ TS(@)$
6 – *stockage tag ac* : $tag \Rightarrow mémoire\ tag(@)$

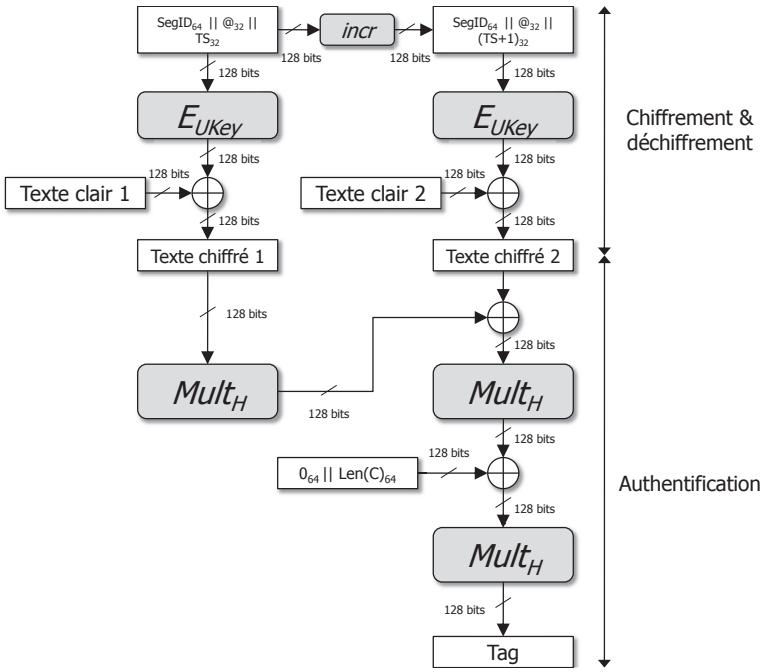

FIGURE 4.5 – Architecture de l'AES-GCM pour un chiffrement authentifié d'une ligne de cache de 256 bits

Algorithme 2 - ***Requête en lecture :***

1 – *chargement horodatage* : $TS \Leftarrow$ *mémoire* $TS(@)$
2 – *chargement tag ac* : *tag* $\Leftarrow$ *mémoire* $tag(@)$
3 – $\{flux\ de\ clé,\ tag\} = AES - GCM\{SegID,\ @,\ TS\}$
4 – *chargement texte chiffré* : *texte chiffré* $\Leftarrow$ *mémoire externe*
5 – *texte clair* = *texte chiffré* $\oplus$ *flux de clé*
6 – *vérification tag ac* : *tag* $\equiv$ *tag ac*
7 – *texte clair* $\Rightarrow$ *mémoire cache*

Le besoin d'unicité des valeurs TS crée un problème si la génération des TS déborde la plage de codage. Une solution typique à ce problème implique le rechiffrement des données stockées avec de nouveaux TS (Yan *et al.*, 2006). Une solution proposant d'utiliser de multiples générateurs de TS à été proposée pour adresser ce problème. Si le compteur TS atteint sa valeur maximum, seule la moitié des données doit être chiffrée de nouveau. Pour notre approche de sécurité, la même idée peut être appliquée basée sur les identifiants des segments. Si un TS associé à un segment dépasse sa capacité, l'ID du segment est incrémenté. Toutes les données incluses dans ce segment sont alors rechiffrées avec cette nouvelle valeur. L'utilisation d'un ID de segment pour la génération du flux de clés aide pour éviter le problème de correspondance des valeurs TS. Si un nouveau chiffrement dû à ce problème a lieu, seule une portion de la mémoire externe est affectée. Bien que non considéré dans l'implémentation courante, le rechiffrement pourrait débuter en arrière plan avant le débordement du compteur où une capacité supérieure pourrait être allouée pour certains TS dépendamment des applications, pour limiter ou éliminer la durée d'inactivité. Pour notre prototype, des TS de 32 bits sont utilisés, indiquant la nécessité de chiffrer à nouveau après 2^{32} écritures de ligne de cache. Cet évènement est supposé arrivé infréquemment.

Le tag de la ligne de cache qui doit être chiffré (étape 2 de l'Algorithme 1) est stocké dans une mémoire AC (étape 6 de l'Algorithme 1). Plus tard, lorsque le coeur de processeur demande une lecture, le tag résultant du ou-exclusif final est comparé avec la valeur AC stockée en mémoire (étape 6 de l'Algorithme 2). Si la donnée est rejouée, relogée ou modifiée, la valeur

récupérée du tag sera différente de la valeur stockée. Une attaque sera alors détectée.

Le stockage requis pour les valeurs AC impacte le niveau de sécurité. Pour limiter ce stockage sans compromettre la sécurité du système, 64 et 128 des bits les plus significatifs du tag sont conservés pour une ligne de cache de 256-bits. Pour un tag de n bits, un adversaire a une probabilité de 1 sur 2^n de modifier avec succès la valeur déchiffrée et de tomber sur la valeur du tag original. La NIST assure la sécurité de chiffrement-authentifié de l'AES-GCM pour des tags de ces tailles (Nat, 2007).

Les tailles des mémoires de stockage des valeurs sur-puce TS et AC représentent une pénalité importante de notre approche qui peut varier largement selon les applications. Ces pénalités sont calculées et analysées pour 4 applications en Section 4.6

4.5 Chargement sécurisé d'application

Comme décrit en Section 4.2, à la mise sous tension ou au reset, le code de l'application doit être chargée de la mémoire flash. Faisant partie du processus de chargement d'application, les contenus de la mémoire flash sont copiés de la mémoire principale par le microprocesseur dés que les registres de celui-ci sont configurés. Pour maintenir la confidentialité et l'authentification du code de l'application, les instructions qui sont stockées en mémoire flash doivent être protégées de façon appropriée par chiffrement et vérification d'authentification. Pour notre système sécurisé, deux scénarii distincts sont considérés :

- **Chargement du code de l'application** : Pour ce scénario, la mémoire SMM est déjà chargée en matériel et seules les instructions de l'application sont chargées dans la mémoire principale. Ce scénario est fréquent si le SMM est inclue dans le bitstream du FPGA.

- **Chargement du code de l'application et de la SMM** : Les informations SMM doivent être chargées dans une mémoire "table" adjacente au microprocesseur et l'application doit être chargée en mémoire principale. Le chargement du SMM est effectué avant le chargement du code

de l'application.

Le détail de ces deux scénarii est maintenant décrit.

4.5.1 Chargement sécurisé du code de l'application

Notre approche pour le chargement sécurisé d'application (Figure 4.6) utilise le même coeur AES-GCM de manière similaire à la technique $Exec_{GCM}$ pour la mémoire principale décrite dans la section précédente. En général, le chargement sécurisé est moins contraignant que la protection de la mémoire principale permettant des optimisations. Puisque les écritures de données et les attaques par rejeu ne sont pas un problème pour la mémoire système flash embarquée, l'horodatage basé sur les segments et les valeurs AC par ligne de cache utilisé pour la protection de la mémoire principale exposent des pénalités non nécessaires. Le besoin de l'adresse et des données du segment aux entrées de l'AES-GCM est alors éliminé. Cette nouvelle politique $Load_{GCM}$ utilise uniquement un TS initial de 32 bits et un IV de 96 bits qui est unique pour chaque application. Ces valeurs remplacent les entrées des blocs E_{Ukey} et *incr* en haut de la Figure 4.5. Excepté pour les clés secrètes AES et les entrées TS et IV, le même circuit AES-GCM utilisé pour la politique $Exec_{GCM}$ est réutilisé pour la politique $Load_{GCM}$ pour le chargement des instructions de l'application.

Algorithme 3 - ***Chargement d'application***

---

1 − *la valeur IV est copiée vers le coeur* AES_{GCM} *avec la politique* $Load_{GCM}$
2 − *la valeur TS est copiée vers le coeur* AES_{GCM} *avec la politique* $Load_{GCM}$
boucle pipelinée (**pour tout le code de l'application**)
3 − *le code de l'application est copié vers le coeur* AES_{GCM} *avec la politique* $Load_{GCM}$
4 − *le code de l'application est déchiffré avec la politique* $Load_{GCM}$
5 − *le code de l'application est chiffré avec la politique* $Exec_{GCM}$
6 − *la donne chiffrée est copiée en mémoire principale*
fin de boucle
7 − *le tag de l'application est comparé avec celui généré par le coeur* AES_{GCM} *exécutant la politique* $Load_{GCM}$. *Si les deux tags correspondent, l'application est chargée de façon sûre et peut être déchiffrée pour une exécution sécurisée*

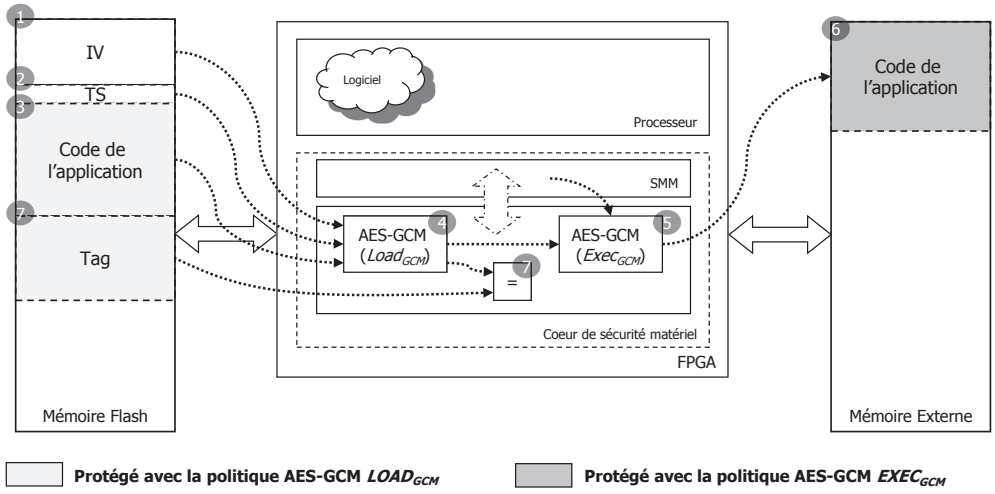

FIGURE 4.6 – Architecture matérielle pour le chargement sécurisé d'applications

Une architecture sécurisée pour le chargement d'application dans un système embarqué (Figure 4.6) utilise les deux politiques $Load_{GCM}$ et $Exec_{GCM}$ pour charger les instructions de la mémoire flash vers la mémoire principale. Ce processus peut être effectué de manière pipelinée avec les multiples coeurs AES 128 bits (pouvant être partagés) qui sont utilisés pour les politiques $Load_{GCM}$ et $Exec_{GCM}$. Comme montré en Figure 4.6, alors que le circuit $Load_{GCM}$ déchiffre les données provenant de la mémoire flash, les opérations d'écriture issues de la politique $Exec_{GCM}$, décrites par l'Algorithme 1, sont appliquées aux instructions avant leur stockage en mémoire principale. Les étapes 3, 4, 5 et 6 de l'Algorithme 3 sont effectuées répétitivement instruction par instruction de façon pipelinée jusqu'à ce que l'application soit chargée. Lorsque le chargement est complété, le tag chiffré de 64 bits placé en mémoire flash est vérifié avec le tag généré par l'AES-GCM exécutant la politique $Load_{GCM}$ pour assurer l'authentification du chargement sécurisé (Étape 7). A l'inverse de la politique $Exec_{GCM}$, qui nécessite de multiples tags stockés en mémoire sur-puce de confiance (1 par ligne de cache protégé), un seul tag de 64 bits est utilisé pour authentifier le chargement de l'application. Ce tag est chiffré et stocké dans une mémoire hors-puce non sécurisée.

4.5.2 Chargement sécurisé du code de l'application et de la SMM

La plupart des plates-formes à base de microprocesseur requièrent la capacité de charger et d'exécuter différentes applications à des moments différents. Ce problème demande non seulement l'initialisation d'une application, mais également de la SMM. Pour les processeurs basés sur FPGA, la flexibilité pourrait être donnée par le chargement de différent bitstreams qui possèdent une configuration SMM alternative pour chaque application. Une autre stratégie est de charger les entrées SMM et faire suivre le chargement du code de l'application.

La configuration SMM pour une application est stockée en mémoire flash sous la forme d'un en-tête d'application (Figure 4.7). Comme le code de l'application, les en-têtes fournissent des informations de sécurité importantes et doivent être protégées. La Figure 4.7 montre la disposition d'un bloc mémoire en flash qui inclut un en-tête d'application protégé et le code de l'applica-

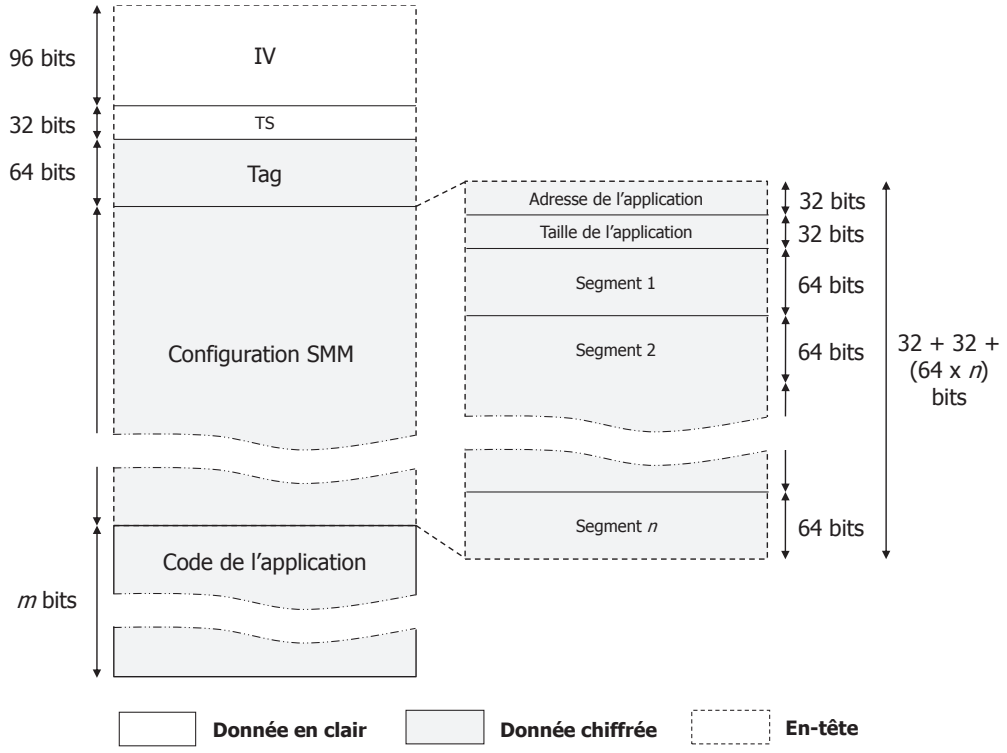

FIGURE 4.7 – En-tête détaillée d'une application sécurisée en mémoire Flash

tion. L'en-tête contient les informations qui sont nécessaires pour effectuer le chargement de l'application, et inclut la configuration SMM, la taille de l'application et l'adresse initiale de chargement. La taille de l'en-tête est variable et dépend de la taille de la configuration SMM. Les composants spécifiques pour l'implémentation de test sont les suivants :

- Un vecteur d'initialisation (IV) de 96 bits.
- Une valeur d'horodatage (TS) de 32 bits.
- Un tag 64 bits de vérification d'authentification pour la configuration SMM.
- La configuration SMM qui contient :
  - L'adresse de base d'application (@) sur 32 bits.
  - La taille de l'application sur 32 bits.
  - 64 bits × le nombre de segments.

Les informations du bloc sont chargées dans la SMM à l'aide de la politique $Load_{GCM}$, et nécessitent donc l'inclusion d'un IV et un TS spécifiquement pour la SMM. A la suite de la configuration de la SMM, les étapes pour le chargement matériel sécurisé d'application sont décrites en Section 4.4, sont effectuées pour compléter le chargement du système. Dans le cas de l'architecture de la politique $Exec_{GCM}$, la taille de la mémoire de stockage des TS, celle de la mémoire des AC et le nombre de segments mémoires supportés par la SMM doivent être suffisamment larges pour stocker les besoins de l'application cible. Cette approche est utilisée pour éviter le développement d'un nouveau bitstream FPGA pour chaque application protégée.

4.6 Approche expérimentale

Un système basé sur FPGA et qui inclut une architecture basée sur le processeur Xilinx Microblaze (Xil, 2009) à été développée pour valider notre approche. Notre coeur de sécurité et les mémoires associées ont été implé-

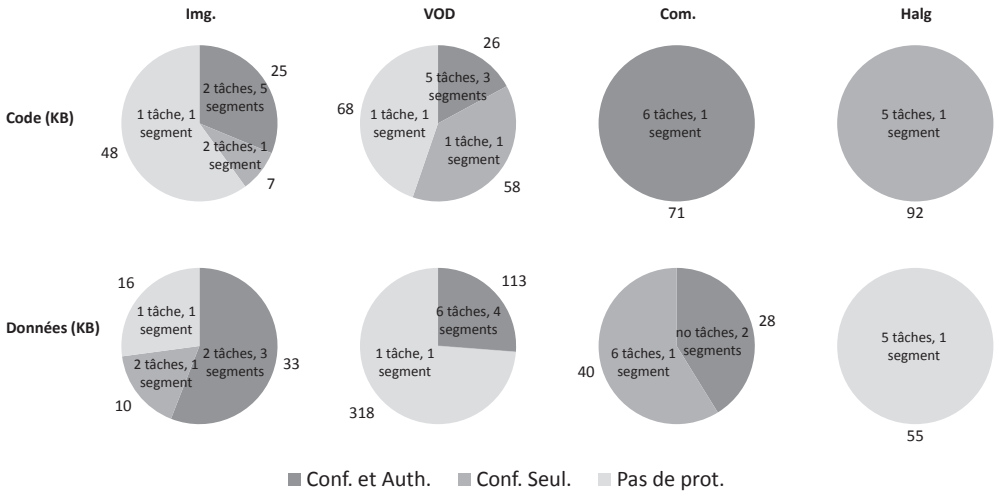

FIGURE 4.8 – Détails de la protection mémoire d'une application par niveau de sécurité

mentés dans la logique et la mémoire embarquée d'un FPGA et interfaçés au processeur via le bus local au processeur de 32 bits (Processor Local Bus, PLB). Dans les différents jeux d'expérimentations, le Microblaze a été accompagné de caches d'instructions et de données de 512 octets puis, par la suite, de 2 kilo-octets. L'OS embarqué MicroC/OS II (LaBrosse, 2002) a été utilisé pour valider notre approche. MicroC/OS II est préemptif et possède un noyau multi-tâches. L'OS peut être configuré par le concepteur pendant la conception de l'application pour n'utiliser que les caractéristiques qui sont nécessaires. Un ordonnancement par priorité est utilisé pour évaluer la tâche qui sera exécutée à un moment précis de l'exécution. Pour explorer l'impact de la gestion de la sécurité sur les performances et la surface, 4 applications multi-tâches ont été utilisées :

- **Image (Img)** : Cette application choisit l'une des deux valeurs d'un pixel et combine les pixel en formes. Les groupes de pixel qui sont trop petits sont supprimés de l'image. Ce processus est basé sur la morphologie mathématique.

- **Vidéo à la demande (VoD)** : L'application inclut une séquence d'opérations pour recevoir des signaux vidéo envoyés chiffrés. Les opérations spécifiques telles que le décodage Reed Solomon (RS), le déchiffrement AES, le décodage moving picture experts group 2 (MPEG) avec correction d'artefact sont employées.

- **Communication (Com)** : Cette application est composée d'une série de tâches utilisées pour envoyer et recevoir des données numériques. Elle inclut les opérations de décodage Reed Solomon, déchiffrement AES et codage Reed Solomon.

- **Algorithmes de hachage (Halg)** : Cette application peut effectuer du hachage cryptographique sélectif basé sur un nombre d'algorithme commun. Les algorithmes supportés sont MD5, SHA-1 et SHA-2.

Les besoins en sécurité des portions des applications ont été estimés sur la base de leur fonction. D'autres options de sécurité sont possibles mais n'ont pas été explorées dans ce travail. Pour l'application Image, les données images et le code de l'application pour filtrer les données sont protégés, mais les données et le code utilisé pour transférer l'application à partir, et vers un système

ne le sont pas. Pour l'application VoD, déchiffrer les données de l'image et les informations spécifiques à l'AES (e.g. la clé de chiffrement) est considéré comme critique. L'algorithme MPEG est également considéré comme propriétaire et son code source est chiffré, tandis que les données MPEG et le code et les données relatives au décodage RS sont laissées non protégées. Pour l'application Com, toutes les données sont considérées comme sensibles et méritent la protection. Dans le but de garantir une opération correcte, le code ne doit pas être changé. La confidentialité et la vérification de l'authentification sont alors appliquées sur tout le code. Les données de l'application sont seulement protégées par la confidentialité. Le code de l'application Halg est aussi chiffré et les données de l'application ne possèdent pas de protection. Par exemple une entreprise pourrait vouloir protéger son code d'une inspection visuelle. Comme la vérification de l'authentification n'est pas souhaitée pour cette application, aucun stockage de TS ou de valeur de tag n'est nécessaire.

La Figure 4.8 résume les tâches, l'utilisation de la mémoire externe, et le nombre de segments pour les applications. Comme noté en Section 4.3, les segments mémoire peuvent être de taille variable. Les 4 applications ont été implémentées avec succès sur une plate-forme SP605 à base de Spartan-6 (Xil (2010a) (Xil, 2010b)) contenant un FPGA XC6SLX45T, 128 mégaoctets (Mo, megabyte, MB) de mémoire externe DDR3 et 32 Mo de mémoire flash. La taille et l'utilisation des mémoires embarquées ont été déterminées après synthèse par l'outil Xilinx Platform Studio 12.2.

4.7 Résultats expérimentaux pour la sécurité de la mémoire principale

Dans un but de comparaison, un système basé sur un Microblaze sans coeur de sécurité a été synthétisé sur un FPGA Spartan-6. Le FPGA XC6SLX-45T contient 43 661 LUTs, 55 576 FFs, 116 BRAMs de 18 kilobit (Kb) et 58 slices pour le traitement du signal (Digital Signal Processing, DSP). Notre configuration de base inclut les caches données et instructions, un timer, un contrôleur de mémoire flash et de mémoire DDR-SDRAM ainsi qu'une interface join test action group (JTAG). Après synthèse avec l'outil Xilinx platform studio (XPS) 12.1, il a été déterminé que cette configuration avec

des caches de 512 octets consomme 3 610 LUTs et 2647 FFs, et opère à une fréquence d'horloge de 75 megahertz (MHz). La même configuration avec 2Ko de cache demande 3335 LUTs et 2538 FFs, et 4 BRAMs supplémentaires. Elle opère à 86 MHz.

Comme exprimé en Section 4.3, la disponibilité du coeur de sécurité permet différents niveaux de sécurité pour différents segments mémoire permettant des compromis sécurité/surface. Dans notre analyse, nous considérons 3 scénarii spécifiques :

- **Pas de protection (NP)** : C'est la configuration de base sans aucune protection mémoire.

- **Protection programmable (PP)** : Le Microblaze et la configuration du coeur de sécurité fournissent exactement la sécurité requise pour chaque segment mémoire de chaque application (Section 4.5).

- **Protection uniforme (PU)** : Le Microblaze et la configuration du coeur de sécurité fournissent le plus haut niveau pour tous les segments mémoire. Comme tous les segments ont le même niveau de sécurité, la taille de la SMM est réduite.

La pénalité en éléments logiques du coeur de sécurité pour la protection programmable n'est pas constante puisque la taille de la SMM dépend du nombre de zones de sécurité définies. Dans le cas d'une protection uniforme, les variations de pénalités résultent des différences dans le circuit de contrôle demandé pour le stockage des tag AC.

4.7.1 Pénalité en surface de la sécurité

Comme montré dans le Tableau 4.4 pour les configurations avec des caches de 512 octets, dans la plupart des cas, la logique requise par le coeur HSC pour une protection programmable est similaire pour une protection uniforme. Le détail de la taille des différentes unités du HSC est donné par le Tableau 4.2 pour la protection uniforme et programmable. Il est à noter que les ressources demandées pour le stockage des tag AC pour la version protection programmable de l'application VoD sont réduites puisque le nombre de tag AC à stocker est réduit. Pour l'application Halg, la vérification de l'au-

Protection Uniforme										
App.	Total		AES_{GCM}		Stockage tag AC		SMM		Ctrl.	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Img.	3485	1122	2065	798	473	154	23	3	924	167
	8.0%	2.1%	4.7%	1.5%	1.1%	0.3%	0.1%	0.0%	2.1%	0.3%
VOD	3619	1149	2065	798	604	177	29	3	921	171
	8.3%	2.1%	4.7%	1.5%	1.4%	0.3%	0.1%	0.0%	2.1%	0.3%
Com.	3470	1121	2065	798	470	153	20	3	915	167
	7.9%	2.1%	4.7%	1.5%	1.1%	0.3%	0.0%	0.0%	2.1%	0.3%
Halg	2576	951	2065	798	0	0	21	3	490	150
	5.9%	1.7%	4.7%	1.5%	0.0%	0.0%	0.0%	0.0%	1.1%	0.3%
Protection programmable										
App.	Total		AES_{GCM}		Stockage tag AC		SMM		Ctrl.	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Img.	3627	1149	2065	798	473	153	214	31	875	167
	8.3%	2.1%	4.7%	1.5%	1.1%	0.3%	0.5%	0.1%	2.0%	0.3%
VOD	3599	1145	2065	798	473	153	161	27	900	167
	8.2%	2.1%	4.7%	1.5%	1.1%	0.3%	0.4%	0.0%	2.1%	0.3%
Com.	3510	1129	2065	798	475	154	61	10	909	167
	8.0%	2.1%	4.7%	1.5%	1.1%	0.3%	0.1%	0.0%	2.1%	0.3%
Halg	2543	949	2065	798	0	0	12	1	466	150
	5.8%	1.7%	4.7%	1.5%	0.0%	0.0%	0.0%	0.0%	1.1%	0.3%

Tableau 4.2 – Détail de l'utilisation logique du coeur de sécurité matériel (HSC)

	Pas de protection	Protection uniforme		Protection programmable	
Arch.	Temps (ms)	Temps (ms)	Pénalité	Temps (ms)	Pénalité
Img. 512	150.5	188.0	24.9%	173.4	15.2%
Img. 2k	131.3	156.9	19.5%	146.9	11.9%
VOD 512	13691.5	16806.4	22.8%	15619.8	14.1%
VOD 2k	11940.3	13751.2	15.2%	13453.5	12.7%
Com. 512	69.1	84.1	21.6%	78.7	14.0%
Com. 2k	60.2	66.7	10.8%	65.4	8.6%
Halg 512	8.6	10.2	18.9%	9.9	15.1%
Halg 2k	7.5	8.7	15.9%	8.6	14.4%

Tableau 4.3 – Temps d'exécution et réduction des performances des applications sécurisées

thentification n'est pas effectuée que ce soit pour la protection programmable ou uniforme, aucune addition matérielle n'est alors demandée. L'utilisation par unité en pourcentage du total des ressources logiques du FPGA disponible est également donnée par le tableau.

L'implémentation de la politique $Exec_{GCM}$ inclut deux unités AES 128 bits avec une seule clé de 128 bits (comme montré en Figure 4.5). Les blocs AES (nommés E_{Ukey} sur cette même Figure) sont implémentés en utilisant une implémentation équilibrée en BRAMs, slices DSP et logiques (Drimer *et al.*, 2010). Bien que non montrés dans le Tableau 4.2, 16 BRAMs et 32 slices DSP sont nécessaires pour les deux coeurs AES 128 bits. La pénalité associée à cette approche est soumise à discussion en Section 4.6.

4.7.2 Pénalité en performance de la sécurité

Le temps d'exécution de chaque application a été déterminé en utilisant les compteurs embarqués dans la logique du FPGA. Le Tableau 4.3 montre le temps d'exécution de chaque application dans chacune des configurations et une évaluation des différentes pertes de performance comparées à la configuration de base. Les expérimentations ont été effectuées pour les trois approches de sécurité avec des caches de 512 octets et de 2 Ko. Le bus PLB 32 bits demande 6 cycles à 75 MHz pour les lectures et les écritures. La latence supplémentaire causée par notre approche de sécurité est de 7 cycles pour une lecture d'une ligne de cache de 256 bits et de 13 cycles pour l'écriture. La pénalité de l'écriture d'une ligne de cache est principalement dûe aux 10 cycles de l'opération AES 128 bits pour la politique $Exec_{GCM}$. La pénalité de lecture est réduite dûe au recouvrement des opérations de la politique $Exec_{GCM}$ et des opérations de lecture du bus. Le pourcentage de perte en performance à cause de la sécurité est plus grand pour les configurations qui incluent des caches plus petits. Ceci est attendu puisque de plus petits caches provoqueront vraisemblablement un nombre d'accès en mémoire plus grand, ce qui augmentera le temps moyen de la latence en lecture. Quelques variations par applications peuvent être vues. Les applications Image et VoD montrent des réductions de performances substantielles (25% et 23% respectivement) avec une protection uniforme même si elles contiennent toutes les deux des segments de données n'étant pas protégés. L'utilisation de la protection programmable permet à ces segments de données d'avoir moins d'impact sur les performances de l'application. Des performances plus modestes (15%

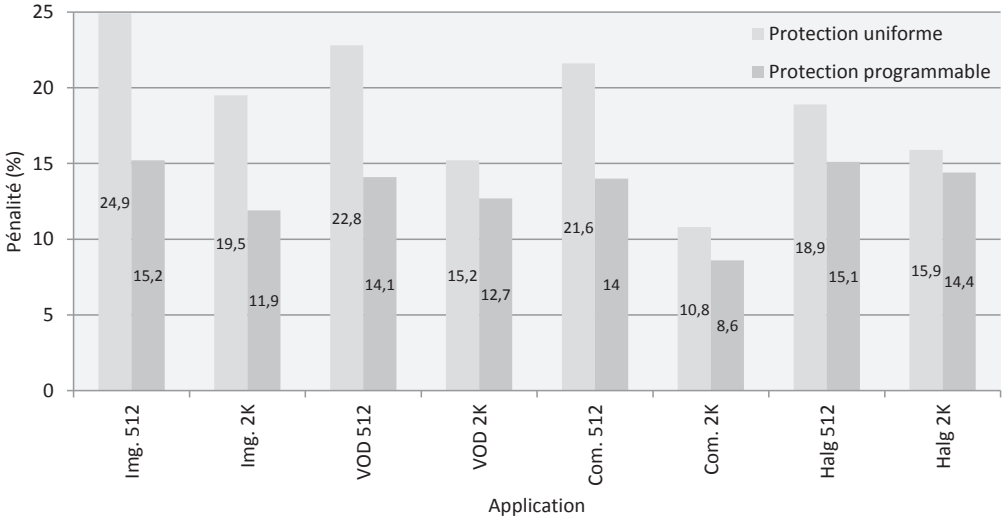

FIGURE 4.9 – Pénalité d'exécution avec les protections uniforme et programmable

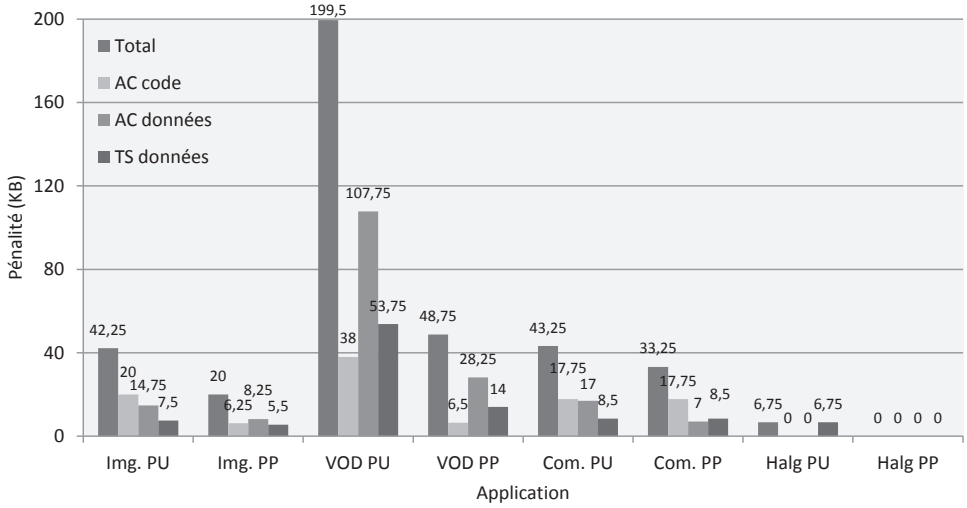

FIGURE 4.10 – Empreinte mémoire des stockages sur-puce TS et AC

et 14% respectivement) sont reportées pour ces configurations. Les effets globaux de nos approches sur les performances sont résumés en Figure 4.9.

Pour la version avec 2 Ko de cache, l'application communication montre une perte en performance pour la version protection programmable de seulement 2% comparée à la version protection uniforme. La Figure 4.8 montre que toutes les données et le code pour cette application doivent être protégés soit par confidentialité et authentification. Le bénéfice de l'aspect programmable est alors limité.

4.7.3 Pénalité en mémoire de la sécurité

Comme dit en Section 4.5, la pénalité en mémoire est le résultat du stockage sur-puce des TS et des tags d'authentification. L'équation 1 fournit les formules pour obtenir l'utilisation de mémoire sur-puce qui sera nécessaire selon les options définies.

Pour notre expérimentation, la ligne de cache est de 256 bits, la taille d'un tag AC est de 64 bits et la taille d'un TS est de 32 bits. En utilisant les valeurs de la Figure 4.8, il est possible de déterminer la taille de la mémoire sur-puce requise basée sur la politique de sécurité sectionnée. Un exemple de calcul de pénalité introduit par les TS et AC est montré pour l'application Image avec programmation programmable. La Figure 4.10 évalue la pénalité de mémoire sur-puce de la sécurité. Comme les valeurs TS et les tags consomment de la mémoire sur-puce embarquée et sécurisée, la disponibilité de ces ressources est réduite pour d'autres applications. Pour l'application VoD, 150 Ko de mémoire sur-puce est sauvé par l'usage de la protection programmable. Ces larges gains proviennent premièrement de la présence de grands segments mémoire qui ne sont pas protégés. Notons que la version programmable de la protection de l'application Halg ne nécessite pas de stockage mémoire puisqu'aucune valeur ne possède de protection par l'authentification et que les valeurs TS ne sont pas nécessaires pour le code de l'application qui est en lecteur seul.

Equations 1 - ***Equations de la sécurité mémoire***

Taille de la mémoire des tag AC pour le code :

$$1 - \textit{pénalité AC} = \frac{\textit{taille tag AC}}{\textit{taille ligne de cache}}$$

	Protection uniforme				Protection programmable			
Arch.	μB + HSC		HSC		μB + HSC		HSC	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
Img. 512	7095	3769	3485	1122	7237	3796	3627	1149
	16.3%	6.9%	8.0%	2.1%	16.6%	7.0%	8.3%	2.1%
Img. 2k	6820	3660	3485	1122	6962	3687	3627	1149
	15.6%	6.7%	8.0%	2.1%	15.9%	6.8%	8.3%	2.1%
VOD 512	7229	3796	3619	1149	7209	3792	3599	1145
	16.6%	7.0%	8.3%	2.1%	16.5%	6.9%	8.2%	2.1%
VOD 2k	6954	3687	3619	1149	6934	3683	3599	1145
	15.9%	6.8%	8.3%	2.1%	15.9%	6.7%	8.2%	2.1%
Com. 512	7080	3768	3470	1121	7120	3776	3510	1129
	16.2%	6.9%	7.9%	2.1%	16.3%	6.9%	8.0%	2.1%
Com. 2k	6805	3659	3470	1121	6845	3667	3510	1129
	15.6%	6.7%	7.9%	2.1%	15.7%	6.7%	8.0%	2.1%
Halg 512	6186	3598	2576	951	6153	3596	2543	949
	14.2%	6.6%	5.9%	1.7%	14.1%	6.6%	5.8%	1.7%
Halg 2k	5911	3489	2576	951	5878	3487	2543	949
	13.5%	6.4%	5.9%	1.7%	13.5%	6.4%	5.8%	1.7%

Tableau 4.4 – Résultats de synthèse des architectures

2 – *code AC* = *total code* × *pénalité AC*

Taille de la mémoire des tag AC pour les données :
3 – *données AC* = *total données* × *pénalité AC*

Taille de la mémoire TS pour les données :
4 – *pénalité TS* = $\frac{TS\ size}{cacheline\ size}$
5 – *données TS* = *total données* × *pénalité TS*

Example pour l'application image avec programmation programmable :
– *pénalité AC* = $\frac{256}{64}$ = 0.25
– *code AC* = $25KB \times\ 0.25 = 6.25\ KB$
– *données AC* = $33KB \times\ 0.25 = 8.25\ KB$
– *pénalité TS* = $\frac{32}{256}$ = 0.125
– *données TS* = $(33KB + 10KB) \times 0.125 = 5.4KB$

4.7.4 Comparaison avec les approches précédentes

En général, les performances de notre approche se comparent favorablement aux approches de sécurités précédentes, montrées par le Tableau 4.1. Bien que AEGIS (Suh *et al.*, 2003b) expose une pénalité de stockage des TS moins important de 6,25%, la pénalité de stockage des valeurs d'intégrité est de 28% ce qui est similaire à notre approche. Une différence significative entre les deux approches est le temps de latence que provoque la vérification de l'intégrité. Alors qu'AEGIS repose sur SHA-1 qui a une latence approchant les 80 cycles, notre nouvelle approche utilise l'authentification du mode AES-GCM qui ne nécessite que 3 cycles pour une entrée de 256 bits. L'approche du moteur de chiffrement parallélisé et de vérification d'intégrité (Parallelized Encryption and Integrity Checking Engine, PE-ICE) (Elbaz *et al.*, 2006) reporte une pénalité en mémoire de 33% et une pénalité en performance de 15%. Ceci pour un niveau de sécurité réduit, contre une attaque par force brute, de $\frac{1}{2^{32}}$. Notre niveau de sécurité contre une attaque de force brute est de $\frac{1}{2^{64}}$ (discuté en Section 4.3), bien que moindre que XOM et AEGIS, est encore approprié pour bien des applications embarquées comme indiqué en Appendice C dans (Nat, 2007). Notre approche utilise et nécessite de la mémoire sur-puce pour le stockage des valeurs AC ce qui peut être un fac-

	Pas de protection	Protection uniforme				Protection programmable			
App.	Temps (ms)	Temps (ms)				Temps (ms)			
		Load	Exec	Total	Pénalité	Load	Exec	Total	Pénalité
Img.	19.84	20.36	1.51	21.87	10.21%	20.36	1.05	21.41	7.92%
VOD	37.32	38.30	2.87	41.17	10.32%	38.30	2.41	40.71	9.07%
Com.	17.46	17.92	1.34	19.26	10.33%	17.92	1.34	19.26	10.33%
Halg	22.48	22.91	1.73	24.64	9.62%	22.91	1.73	24.64	9.62%

Tableau 4.5 – Temps du chargement et de l'exécution sécurisée des applications

	En-tête de l'application et SMM
Application	Taille (octets)
Img.	128
VOD	112
Com.	64
Halg	48

Tableau 4.6 – Pénalité flash des en-têtes pour les applications

teur limitant pour les plates-formes embarquées où la plupart des segments doivent être protégés. XOM, PE-ICE, Yan-GCM et AEGIS permettent une combinaison de stockage des informations sécurisées sur-puce et hors-puce. Cependant, l'extension récente de la mémoire sur-puce disponible sur les FPGAs, limite l'impact de ce problème. Le niveau de sécurité contre les attaques par force brute de notre approche peut être doublé ($\frac{1}{2^{128}}$) si le stockage sur-puce des AC est également doublé, bien que cette option n'a pas été explorée dans ce travail.

4.8 Résultats expérimentaux pour la sécurité de chargement d'application

Les résultats suivants considèrent le coût en surface et en performance en lien avec le chargement sécurisé et l'exécution d'une application donnée. Les deux cas de chargement considérés en Section 4, (chargement du code de l'application seulement et chargement du code de l'application et de la mémoire SMM), sont décrits pour des systèmes nécessitant de la protection uniforme et programmable.

4.8.1 Coût en latence du chargement du code de l'application à partir de la mémoire flash

La carte Xilinx décrite en Section 4.5 a été utilisée pour valider notre approche sécurisée. Pour notre prototype, le code de l'application est lu depuis la flash, déchiffré avec la politique $Load_{GCM}$, et chiffré de nouveau avec la politique $Exec_{GCM}$. La performance et le coût en mémoire flash de la politique $Load_{GCM}$ varient selon les applications. Pour cette expérience, toutes les lectures et opérations de la mémoire flash prennent place à 85 MHz. Un total de 580 cycles est demandé pour la lecture de 256 bits d'une donnée stockée en mémoire flash, par morceau de 8 bits. Le Tableau 4.5 montre le détail du temps nécessaire pour effectuer le chargement de l'application de la flash jusqu'à ce que le code soit envoyé en mémoire DDR-SDRAM. Les résultats sont montrés sans protection, avec protection uniforme et avec protection programmable pour les 4 conceptions évaluées en Section 6. Le délai de lecture du code présent dans la mémoire flash est montré dans le cas ou aucune protection n'est appliquée. La politique de déchiffrement $Load_{GCM}$

et les délais du chiffrement $Exec_{GCM}$ sont également montrés pour les deux autres cas. Dans notre implémentation, les mêmes coeurs AES sont multiplexés entre les opérations de $Load_{GCM}$ et $Exec_{GCM}$, et deux clés secrètes de 128 bits distinctes sont utilisées, une pour chaque politique. Un total de 1 024 LUTs est nécessaire pour le contrôle de $Load_{GCM}$ et le multiplexage de l'AES-GCM. Le temps combiné d'une considération en pipeline de ces étapes est limité par le temps d'accès à la flash. Le temps de chargement de l'application est alors grossièrement équivalent au temps de chargement de la mémoire flash. Il y a une pénalité de 9% en temps de chargement en moyenne pour les applications dûes au besoin de protection de la mémoire.

4.8.2 Coût en délais du chargement du code de l'application et de la mémoire à partir de la mémoire flash

La pénalité en surface demandée pour le chargement d'une mémoire SMM spécifiquement à une application est minimale en comparaison du coût de la sécurisation de la mémoire. La mémoire flash supplémentaire requise pour maintenir les informations d'en-tête (la configuration SMM étant inclue), pour chaque application est montrée dans le Tableau 4.6. Comparé aux applications ciblés, le temps de chargement et du déchiffrement pour ces informations complémentaires est négligeable. Comme la SMM doit être inscriptible pour supporter la configuration, une mémoire interne au FPGA BRAM de 18 Kb est utilisée pour maintenir les 15 segments mémoires pour l'application la plus large de notre suite de tests. Le nombre de LUTs pour le contrôle du module SMM passe de 214 (Table 4.2) à 252.

4.9 Conclusion

Dans ce chapitre nous avons présenté une approche de sécurité pour la mémoire externe dans un système embarqué. L'approche fournit une implémentation avec faible pénalité, et autorise le chiffré authentifié basé sur un standard approuvé par la NIST, AES-GCM. L'approche minimise la logique requise pour l'authentification et prend avantage de la forte augmentation des mémoires embarquées dans les FPGAs. Le chiffrement et l'authentification sélectifs pour différentes parties d'une application ne demandent pas d'instructions microprocesseur supplémentaires, ni des modifications drastiques

du système d'exploitation. Les bénéfices de notre coeur de sécurité sont démontrés et quantifiés avec 4 applications embarquées implémentées sur un FPGA Spartan-6. La taille et les pénalités en performance du circuit pour supporter le chargement sécurisé d'application sont aussi quantifiées.

Chapitre 5

La Reconfiguration Dynamique au Service de la Sécurité

Contenir la mer de multi-standards cryptographiques est un vrai défi pour tout concepteur puisqu'il en résulte une densification du silicium. Depuis peu, la reconfiguration dynamique partielle, a su répondre en partie à cette problématique : un composant logique peut être échangé et remplacé par un autre à la volée ce qui limite par conséquent la surface. Dans ce chapitre, nous nous intéressons à la proposition d'une approche de bout-en-bout pour la reconfiguration dynamique partielle, où nous présentons une hiérarchie de dépôts de bitstreams pour les systèmes reconfigurables. Elle est basée sur trois niveaux spécifiques, autorisant le téléchargement de bitstreams partiels depuis des serveurs distants lorsqu'un portfolio de bitstreams est nécessaire. Ce chapitre est quelque peu décorrélé des notions de sécurité entrevus précédemment. Pourtant, comme nous le verrons, la cryptographie possède des applications directes et peut s'appuyer fortement sur la reconfiguration partielle et sur les architectures ici proposées.

Ce chapitre revient sur les propriétés de la reconfiguration dynamique partielle dans une première partie. Suivant l'introduction, les parties deux, trois et quatre sont organisées de la manière similaire et évoquent notre approche autour de différents niveaux de hiérarchie. Ces parties incluent les architectures et les résultats sur plate-forme de prototypage. Enfin, la partie cinq clôt ce chapitre en y apportant les conclusions.

Sommaire

5.1 Reconfiguration dynamique partielle

Les FPGAs fournissent des SoCs reconfigurables avec possibilité de construire des systèmes à la demande. Un seul FPGA reconfigurable pour beaucoup d'applications est une bonne réponse aux problèmes critiques des conceptions ASIC. L'explosion des coûts de conception et de production est dûe à la demande continuelle d'augmentation de la densité des technologies semiconducteur et à la difficulté de mettre à jour et corriger à la fois des firmwares matériels et logiciels. De plus, les blocs dur FPGAs comme les processeurs, les mémoires, les DPSs et les interfaces de communication haute vitesse apportent une grande flexibilité aux niveaux matériels et logiciels, à gros ou fin grain.

Dans l'industrie de la télécommunication, les terminaux universels reconfigurables sans-fil sont maintenant des idées bien connues qui sont d'abord apparues dans le domaine militaire avant de devenir civilement populaire dans les années 90. Ce sujet "chaud" est une conséquence directe des performances des FPGAs. Cette technologie donne accès à un parallélisme massif, fournit suffisamment de puissance de calcul pour réaliser des frontaux numériques (Digital Front End, DFE) et la possibilité d'être reconfiguré avec une consommation en puissance modérée (Becker *et al.*, 2003). En supposant qu'un dispositif devrait supporter plusieurs services numériques de téléphonie mobile, des services de diffusions digitales, et/ou des services de transferts de données numériques, il peut s'appuyer sur la reconfiguration partielle. Les dispositifs actuels imposent un nombre limité de services à cause de la non flexibilité des parties analogiques mais ceci tend à être évité par la radio logicielle (Software Defined Radio, SDR). Il s'agit d'un ensemble de techniques qui permettent la reconfiguration d'un système de communication sans changer physiquement les éléments matériels. Le but sous-jacent est de produire des appareils capables de supporter différents services (multi-standard) avec une adaptation de leurs composants matériels en fonction du réseau sans fil comme le système global mobile (Global System Mobile, GSM), le service radio général de paquet (General Packet Radio Service, GPRS), le système de télécommunication universel et mobile (Universal Mobile Telecommunications System, UMTS) et l'accès intéropérable mondial aux ondes radio (Worldwide Interoperability for Microwave Access, WIMAX). De plus, ils doivent être capables de gérer les standards réseaux comme IEEE 802.11 plus connu sous le nom de Wi-Fi. Delahaye et al. (Delahaye *et al.*, 2004) montre

la faisabilité de la reconfiguration dynamique partielle sur une plate-forme radio logicielle hétérogène qui fournit une approche flexible pour concevoir des systèmes hautement réutilisables à la demande.

De tels dispositifs requièrent de s'adapter dynamiquement à un sous-ensemble de leurs fonctions pour prendre en considération toutes les variations en "temps-réel". Ils peuvent donc utiliser la reconfiguration dynamique partielle (RDP, Dynamic Partial Reconfiguration, DPR) en échangeant les ressources matérielles à la demande.

La reconfiguration des FPGAs Virtex de Xilinx peut être exploitée de différentes manières, partiellement ou globalement et de façon externe (exo-reconfiguration) ou interne (endo-reconfiguration). Dans ce contexte l'aspect reconfiguration dynamique et partielle des Virtex demande des ressources supplémentaires pour stocker les nombreux bitstreams partiels. A l'heure actuelle, les chercheurs exploitent les mémoires flash locales comme dépôt de bitstreams et les serveurs de fichiers sont accédés par des protocoles standards comme le protocol de transfert de fichiers (File Transfer Protocol, FTP) ou le système de fichier en réseau (Network File System, NFS). Parce que les mémoire sont des ressources rares dans les systèmes embarqués faible-coût et haut-volume, nous faisons face à une migration des mm^2 du FPGAs vers les mémoires. Bien que les mémoires faibles-coût sont en faveur de cette migration, il subsiste des désavantages :

- Leur réutilisation peut être extrêmement faible, puisque ces mémoires ne peuvent être utilisées qu'une seul fois par exemple lors de la mise sous-tension.

- L'équilibre en terme de mm^2 de silicium, réduction du nombre de composants, de surface des circuits imprimés (Printed-Circuit-Board, PCB), de consommation en puissance et temps moyen entre pannes (Mean Time Between Failure, MTBF) est négatif.

- Pour une seule fonction à implémenter, l'espace possible de bitstreams peut être immense et devenir plus grand que les mémoires locales. Trois facteurs sont en partie responsable :

  - Les familles de FPGAs avec les nombres grandissant de dispositifs, leurs tailles variables, packages et variations de grade de vitesse.

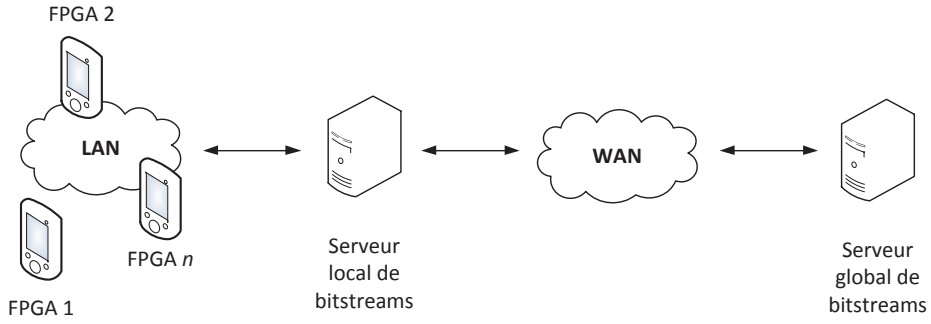

FIGURE 5.1 – Architecture d'un réseau LAN/WAN

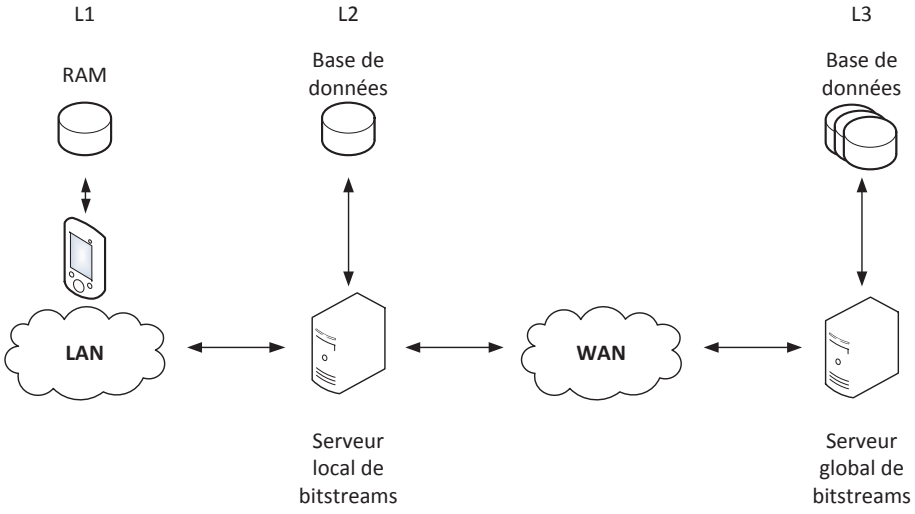

FIGURE 5.2 – Niveaux L1, L2 et L3 de la Hiérarchie de dépôt de bitstream

- Le nombre des configurations possibles, malheureusement dépendant de caractéristiques spatiales comme la forme et la surface de positionnement des IPs.

- La durée de vie naturelle des IPs commerciales produisant régulièrement des nouvelles versions et mises-à-jours.

Une hiérarchie de dépôts de bitstreams devient alors nécessaire et doit communiquer, à travers des canaux physiques et des protocoles réseaux adaptés, avec les FPGAs reconfigurables partiellement. Tous les concepteurs de système FPGA veulent les meilleures performances au coût le plus faible pour télécharger les bitstreams partiels dans le FPGA. Un chargement à partir d'une mémoire locale proposera une latence faible avec une capacité de stockage faible, à l'instar d'un accès à un serveur distant où la latence d'accès sera irrémédiablement plus élevée mais où la capacité de stockage sera bien plus grande. Une hiérarchie de dépôts de bitstreams délivre toutes les versions d'une IP à tout le portfolio de FPGAs cibles. Pour une topologie réseau typique (Figure 5.1), cette hiérarchie est composée de trois niveaux (Figure 5.2) :

- **L1** : Un cache mémoire local de bitstreams.

- **L2** : Un serveur rapide de bitstreams localisé dans un LAN dédié qui utilise un protocole simplifié.

- **L3** : Un serveur plus lent, standard, qui peut être localisé n'importe où et accédé via des protocoles comme le protocole à transmission de contrôle (Transmission Control Protocol, TCP) ou le protocole avec datagramme utilisateur (User Datagram Protocol, UDP).

Dans les lignes qui suivent, nous présentons et dérivons chaque niveau en terme logiciel, matériel et de protocoles de communication. Nous fournirons une spécification et une optimisation d'implémentation d'une couche minimale logicielle abstrayant l'accès aux ressources matérielles impliquées.

5.2 Niveau de hiérarchie L1

Le niveau L1 est le niveau carte où les concepteurs assemblent les FPGAs et les mémoires RAM (Random Access Memory). Les bitstreams peuvent être stockés dans les mémoires et il est très commun de voir utiliser des tailles de 512 Mo. C'est la manière la plus populaire pour stocker les bitstreams et construire des prototypes parce qu'il y a un grand nombre de cartes d'évaluation avec des lecteurs flash à des bas-coûts pour les universités et les chercheurs. Mais moins de mémoire il y a, moins cher est le système à produire en grand volume. Le niveau L1 est géographiquement le dépôt le plus proche du FPGA, et celui avec la latence la plus faible. Sa latence dépend, bien évidemment, des mémoires et des bus employés. Il sera toujours le meilleur comparé avec des équipements réseaux.

La communauté de la reconfiguration partielle (RP, partial reconfiguration, PR) s'accorde sur le fait que, dans les domaines applicatifs avec des contraintes temps réels fortes, la latence RP est l'aspect le plus critique d'une implémentation. Si celle-ci ne peut être suffisamment réduite, l'intérêt de la RP dans la constitution de systèmes efficaces peut être compromis. Les temps de reconfiguration seront hautement dépendant de la taille et de l'organisation des régions partiellement reconfigurables du FPGA. Les Virtex-2 par leur organisation en colonne produisent des bitstreams partiels plus larges que nécessaire. Les Virtex-4 et Virtex-5 ont relâché cette contrainte et acceptent maintenant des régions avec des formes arbitraires. Ils ont des régions composées de 41 mots de 32 bits. Le plus petit Virtex-4, le LX15, possède 3740 régions, et le plus large, le FX140, 41152. Selon les documentations Xilinx, quatre méthodes de reconfiguration existent et ont des vitesses de téléchargement différentes :

- Externe au FPGA (exo-reconfiguration) :
  - port de configuration série, 1 bit, 100 MHz, 100 megabit/s (Mb/s).
  - port boundary scan (JTAG), 1 bit, 66 MHz, 66 Mb/s.
  - port SelectMap, 8 bits parallèle, 100 MHz, 800 Mb/s.
- Interne au FPGA (endo-reconfiguration) : avec le port interne de confi-

guration (Internal Configuration Access Port, ICAP) (Xilinx, 2006), Virtex-2, 8 bits parallèle, 100, MHz, 800 Mb/s.

Bien sûr, ces valeurs crêtes sont uniquement objectives et les ports ICAP des Virtex-4 et Virtex-5 ont des formats d'accès plus grands (mots de 16 et 32 bits respectivement). Dépendamment des capacités du concepteur du système à construire un pipeline de données efficace du stock de bitstreams (RAM, flash ou distant) jusqu'à l'ICAP, les performances seront plus ou moins proches de ces valeurs crêtes. Les deux questions importantes ici sont "quelle latence est acceptable" pour une application donnée et "quels sont les coût liés" en terme de coût système (mémoires ajoutées, composants périphériques). Particulièrement, dans le cadre de la reconfiguration partielle, pour être capable de comparer les contributions, nous devons identifier la taille moyenne d'un bitstream partiel et la latence moyenne "acceptable" de reconfiguration. Enfin, parce que les systèmes peuvent s'exécuter à différentes fréquences, il est nécessaire d'intégrer la fréquence système dans ces nombres. Les séries Virtex-2, Virtex-4, Virtex-5 et Virtex-6 contiennent au moins un port ICAP qui peut être interfacé avec des IPs matérielles et des coeurs de processeurs synthétisables (Microblaze, OpenRISC, LEON) ou en dur (PowerPC, ARM). Le débit de téléchargement annoncé par Xilinx en mode reconfiguration interne, est bridé à 800 Mb/s lorsque les accès sont effectués par mots de 8 bits. Le coût de son implémentation est de 150 slices et 1 BRAM.

5.2.1 Architecture du cache

L'utilisation de mémoires flash par l'intermédiaire de cartes de stockage de masse ou de mémoires intégrées sur-puce, est bien connue et utilisée, ou moins lors du boot. Ce type de stockage non volatile est utile pour maintenir une large gamme de bitstreams lorsque le temps d'accès n'est pas une contrainte. Sans parler des temps de transactions, la lecteur d'un mot de 32 bits est proche des 500 cycles d'horloge. Cette valeur est bien sûr dépendante de la technologie flash et du contrôleur associé. Avec l'utilisation d'un cache, les concepteurs sont capables de résoudre ce "problème" en copiant un bitstream dans une mémoire à accès plus rapide qui est localisée plus proche du CPU. Le problème ici est que les bitstreams partiels ont une taille de plusieurs centaines de Ko lorsqu'aucune technique de compression est utilisée (Huebner *et al.*, 2004b). Les mémoires BRAMs ne sont alors pas la réponse

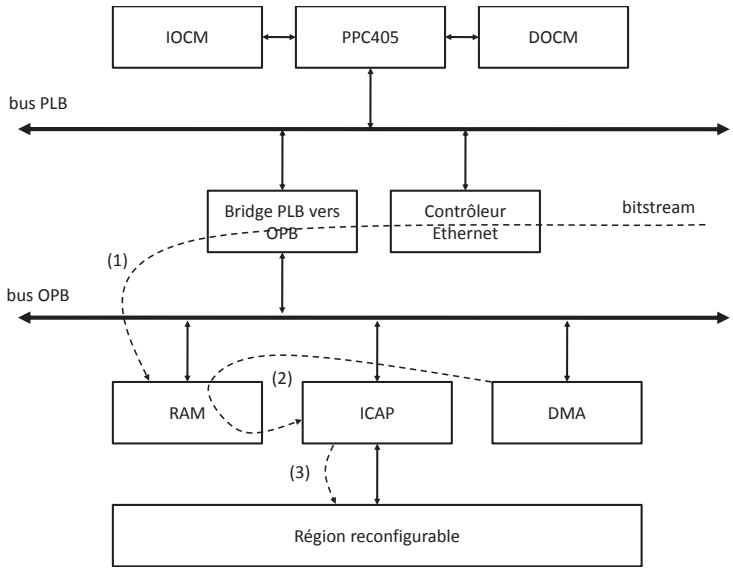

FIGURE 5.3 – Architecture matérielle du niveau L1

puisque leur nombre total disponible est faible comparativement. Avec cette petite et simple analyse en tête, nous proposons l'utilisation d'une mémoire SRAM comme mémoire cache. C'est un compromis entre la mémoire volatile plus rapide (BRAM) et la mémoire non-volatile plus lente (flash). Ce cache complètement décrit en logiciel est efficace pour accélérer le temps de reconfiguration de quelques bitstreams critiques avec un bas-coût en mémoire. Bien sûr, ce temps pourrait être d'autant plus réduit qu'un IP matériel pourrait être développé pour effectuer le même travail. Pour éviter une implémentation en logique complexe, le cache est pleinement associatif ce qui signifie que chaque ligne de la mémoire principale ne peut être enregistrée qu'à une seule adresse de la mémoire cache. La politique de remplacement de cache retenue est la politique qui remplace la ligne la moins récemment utilisée (Least Recently Used, LRU). Si l'espace mémoire vient à manquer dans la mémoire cache pour stocker tous les bitstreams, le bitstream le moins utilisé est remplacé par ceux les plus utilisés.

La stratégie de produit Xilinx a toujours été en faveur de la réduction de la taille des bitstreams partiels. Depuis la famille Virtex-4, une reconfiguration partielle 2D peut être appliquée et la contrainte en colonne des Virtex-2 n'est plus un goulot d'étranglement. De ce fait, la taille moyenne des bitstreams partiels est plus petite. Cependant, plus la ligne de cache est petite, le moins d'espace sera perdu lors du chargement de bitstreams pour lesquels leur taille n'est pas un multiple pur de la longueur de la ligne de cache. Une défragmentation de cet espace peut être fait "hors-ligne" lorsqu'aucun transfert de bitstreams est effectué. Cette technique n'est pas détaillée ici, parce qu'elle n'a aucune influence directe sur les latences de téléchargement.

5.2.2 Architecture matérielle

Tout le système est bâti autour de versions d'outils matures pour utiliser la reconfiguration dynamique partielle. Ainsi les outils Xilinx de développement (embedded development kit, EDK) et ISE 9.2, et PlanAhead 10.1 ont été utilisés. L'architecture matérielle (Figure 5.3) se base sur un FPGA Virtex-2 Pro 30 cadencé à 100 MHz sur une plate-forme d'évaluation pour les universités (Xilinx University Program, XUP) de Xilinx. Un processeur Power PC PPC405 exécute le logiciel de RP. Nous considérons que les IPs dynamiques communiquent avec l'environnement du FPGA directement via quelques PADs. Alors, le FPGA est équivalent à un ensemble de composants

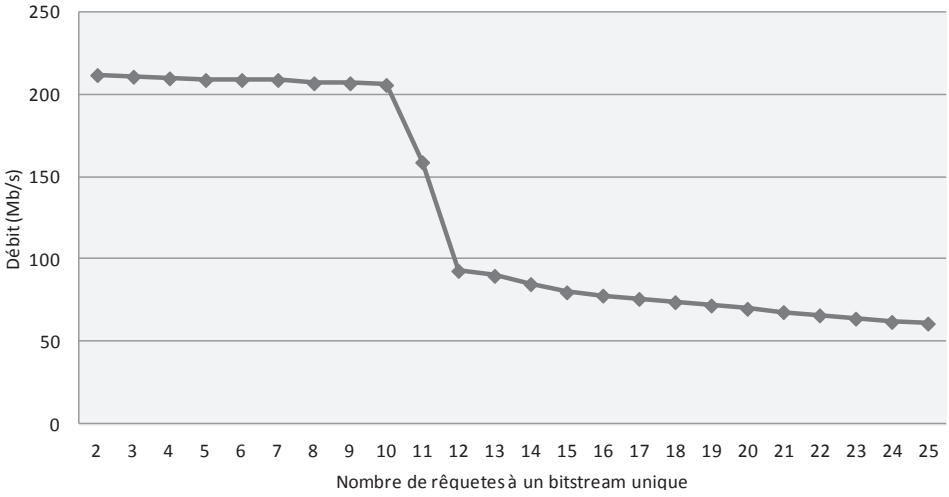

FIGURE 5.4 – Débit de la reconfiguration partielle

reconfigurables capables d'interchanger rapidement une fonction avec une autre. La communication avec le PPC405 et les communications inter-IPs ne sont pas étudiées dans ce chapitre mais peuvent être implémentées avec les bus macro Xilinx et de Huebner (Huebner *et al.*, 2004a), et des wrappers pour les périphériques sur-puce (On-Chip Peripheral Bus, OPB) et PLB aussi bien qu'avec des crossbars externes comme dans (Bobda *et al.*, 2005). Le PPC405 possédant une architecture Harvard, nous ajoutons deux mémoires pour stocker le code de l'application et des données, respectivement appelées mémoire sur-puce d'instruction (Instruction On Chip Memory, IOCM) et mémoire de données sur-puce (Data On-Chip Memory, DOCM). Le PPC405 communique avec ces périphériques à travers deux bus connectés par un bridge : le bus PLB pour les plus rapides et le bus OPB pour les plus lents. Le contrôleur Ethernet est connecté au PLB et utilise un accès direct en mémoire (Direct Memory Access, DMA) intégrée pour accélérer le transfert des paquets reçus vers la mémoire cache localisée en mémoire externe. Un second composant DMA est instancié et dirigé par le PPC lui-même pour copier le bitstream du cache vers le composant ICAP pour éviter les latences de transfert du PPC405. Enfin, l'ICAP, connecté au bus OPB, gère les accès et le télé-chargement de bitsteams vers les régions reconfigurables. Une reconfiguration externe complète est effectuée à la ré-initialisation via le lien JTAG tandis que la reconfiguration interne est faite dynamiquement au travers du port ICAP.

5.2.3 Résultats

Nos mesures sont basées sur l'endo-reconfiguration répétitive d'IPs cryptograhiques comme AES, DES et 3DES. Pour obtenir les résultats suivants (Figure 5.4), le cache est configuré pour stocker 16 lignes de cache de 32 slots de 1496 octets. Si l'on considère un bitstream partiel de 74 Ko, il est possible de stocker 10 bitstreams. Le côté gauche de la courbe montre le débit obtenu lorsque tous les bitstreams sont présents en cache. Cette courbe montre un débit moyen de téléchargement de bitstreams du cache vers l'ICAP d'environ 2.1 Mb/(s.Mhz) soir 210 Mb/s lorsque le PPC405 est cadencé à 100 MHz. Comme le cache n'est capable de stocker "que" 10 bitstreams, le débit est dramatiquement réduit lorsque le nombre de bitstreams demandé est plus élevé que la capacité du cache. Côté droit de la figure, nous pouvons souligner que plus il y a de nombre unique de requête pour l'obtention d'un bitstream, mois le débit est important (50 Mb/s). Lorsqu'aucun bitstream n'est trouvé dans

le cache local, le niveau L2 de la hiérarchie est automatiquement interrogé. Ce niveau est capable de donner accès à un grand nombre de bitstreams partiels par l'intermédiaire d'un serveur local.

5.3 Niveau de hiérarchie L2

Le niveau L2 est le niveau LAN avec un protocole au niveau lien de données spécifiques. Il peut fournir un service de reconfiguration avec une latence moyenne de 10 millisecondes (ms). Ethernet, dans son usage le plus simple, est un médium partageant des mécanismes par lesquels bien des protocoles ont été construits, et peut être vu comme un excellent lien série. En terme de coût d'achat et de facilité de déploiement, c'est un candidat premier pour le transfert de bitstreams entre des dispositifs proches comme notre FPGA et un serveur de bitstreams au sein d'un LAN.

Pas seulement dédiée à la RDP, la note d'application Xilinx (Xilinx, 2006), décrit un système construit autour d'un Virtex-4 FX12 cadencé à 100 MHz. Il contient un processeur Microblaze synthétisé exécutant le code d'un serveur avec protocole de transfert hypertexte (Hypertext Transfer Protocol, HTTP). Ce serveur télécharge des fichiers via un LAN 100 Mb/s Ethernet. La pile de protocoles utilisée est la Dunkel lwIP (Dunkels, 2001) et le système d'exploitation basé sur le micro-kernel Xilinx (Xilinx MicroKernel, XMK). Une mémoire externe de 64 Mo est nécessaire pour stocker les tampons de lwIP. Le débit annoncé est de 500 Ko/s, soit 40Kb/(s.MHz). Ce chiffre est 200 fois inférieur à celui de l'ICAP. (Lagger *et al.*, 2006) proposent un système (Reconfigurable Object for Pervasive Systems, ROPES) dédié à l'accélération de fonctions cryptographiques. Il est construit avec un Virtex-2 1000 tournant à 27 MHz. Le processeur est le Microblaze synthétisé exécutant le code du système d'exploitation μCLinux. Le téléchargement des bitstreams s'effectue via Ethernet avec les protocoles HTTP et FTP au-dessus d'une pile TCP/IP. Les bitstreams ont un taille moyenne de 70 Ko, et les latences de RDP sont d'environ 230 ms avec HTTP et environ 1200 ms avec FTP. La vitesse de reconfiguration est comprise entre 30 et 60 Ko/s, soit un maximum de 17 Kb/(s.Mhz). Williams et Bergmann (Williams et Bergmann, 2004) propose μCLinux comme plate-forme de RDP universelle. Ils ont développé un pilote au-dessus de l'ICAP. Ce pilote active le téléchargement de bitstreams provenant de n'importe quelle location grâce à la séparation complète entre

les accès à l'ICAP et au système de fichier. La connexion entre un système de fichier distant et l'ICAP est effectuée au niveau utilisateur par une commande shell ou un programme utilisateur. Lorsqu'un fichier système distant est monté via NFS/UDP/IP/Ethernet le bitstream peut être téléchargé dans la région reconfigurable. Le système est construit autour d'un Virtex-2 et le processeur exécutant le système d'exploitation est un Microblaze. Les auteurs acceptent le fait que la facilité de fonctionnement a un coût en terme de performances. Aucune mesure n'est fournie avec l'étude mais nous avons effectué des mesures dans un contexte similaire pour palier à ce défaut. Une rapidité de transfert s'étalant de 200 Ko/s à 400 Ko/s a été mesurée, ce qui représente une performance maximale d'environ 32 Kb/(s.Mhz).

Cet état de l'art établit que la solution "Microblaze + Linux + TCP" domine. Malheureusement, les débits de téléchargement sont bien inférieurs à la bande passante de l'ICAP et du réseau. De plus, les besoins en mémoire sont de plusieurs megaoctets, ce qui demande l'addition de mémoires externes. Linux et sa pile TCP/IP ne peuvent s'exécuter sans une mémoire externe pour stocker le code du noyau et les tampons des protocoles de communication. Deuxièmement, l'implémentation, et probablement la nature (spécifiée dans les années 80 pour des liens plus lents et moins fiables) des protocoles, sont telles que seulement quelques centaines de Kb/s sont atteignables dans un LAN traditionnel. L'empreinte mémoire excessive et les protocoles mal ajustés sont des problèmes que nous avons l'intention de réduire.

5.3.1 Lien de données Ethernet 100 Mb/s

Le protocole réseau Ethernet IEEE 802.3 (Ethernet), créé par Metcalfe et Boggs au Parc Xerox dans les années 70, est un ensemble riche de technologies de communication pour connecter à faible coût des ordinateurs entre eux. Il est basé sur la diffusion de paquets sur un médium partagé avec detection de collision (Carrier Sense Multiple Access/Collision Detection, CSMA/CD). L'insertion de commutateurs et de concentrateurs pour simplifier le câblage et pour améliorer la vitesse et la qualité de services, transforme le LAN en un ensemble de liens point-à-point connectés à travers d'un équipement de routage. Avec cette topologie, deux équipements connectés au même commutateur communiquent avec un lien quasi-privé (exceptée la diffusion broadcast de paquets). L'évolution d'Ethernet est telle que des dizaines, et même des centaines de Mb/s sont maintenant disponibles à très bas-coût avec des

n	p	$(1-p)^n$	$1-(1-p)^n$
10^5	10^{-9}	0.9999	10^{-4}
10^5	10^{-10}	0.99999	10^{-5}
10^5	10^{-11}	0.999999	10^{-6}
10^6	10^{-9}	0.999	10^{-3}
10^6	10^{-10}	0.9999	10^{-4}
10^6	10^{-11}	0.99999	10^{-5}

Tableau 5.1 – Estimation des taux d'erreurs d'un téléchargement de bitstreams

taux d'erreurs quasiment nuls. Avec notre hiérarchie de dépôts, le serveur de bitstreams est connecté au même LAN que notre système. Aucun routage IP comme le protocole de contrôle de messages Internet (Internet Control Message Protocol, ICMP), le protocole de résolution d'adresse (Address resolution protocol, ARP), TCP ou UDP n'est alors nécessaire. L'inconvénient immédiat est que cela ne permet pas le télé-chargement de bitstreams d'une autre machine par Internet, mais il est bon de souligner que ce sera la fonction du niveau L3.

Pour caractériser la vitesse de cette topologie LAN, un test a été effectué pour définir à quelle vitesse maximale les paquets Ethernet pourraient être envoyés d'un PC à la carte. Une application envoie les paquets aussi rapidement que possible, sans protocole spécifique, flot de contrôle et détection d'erreurs. L'accès direct au contrôleur Ethernet au niveau MAC peut être fait simplement grâce aux sockets raw Linux. Ce test démontre que la limite en vitesse de 100 Mb/s est atteignable. Ce débit dépend uniquement du PC et des performances de commutation. L'absence d'erreur de transmission durant des semaines de tests montre que, dans ce contexte, la qualité du lien de données est telle qu'il n'est probablement pas nécessaire d'implémenter une detection d'erreurs complexes.

5.3.2 Taux d'erreurs

Considérons les tailles des bitstreams partiels d'un maximum de 100 Ko soit 800 Kb. Chaque bit transmis a une probabilité p d'être erroné. Avec les concentrateurs actuels ces taux d'erreurs sont très petits, de magnitude 10^{-9}. La probabilité d'envoyer n bits avec une erreur est obtenue par la formule $(1-p)^n$, et le taux d'erreur est $1-(1-p)^n$ et donne les valeurs du Tableau 5.1.

Ceci montre que les taux d'erreurs sont très bas pour les bitstreams. Ils sont en faveur d'une détection d'erreurs très simple et d'une stratégie de récupération basique : une reprise au niveau bitstream plus qu'au niveau paquet. C'est pourquoi le protocole lien de données décrit dans (Bomel *et al.*, 2008) est un bon candidat pour de telles communications de données. De plus, lorsque des perturbations externes arrivent, elles seront concentrées sur quelques paquets adjacents et les bits erronés auront une corrélation plus forte. Parce que nous ne somme pas dans le cadre de transmissions radios, nous n'avons pas besoin de décorréler ces erreurs avec des entrelaceurs type

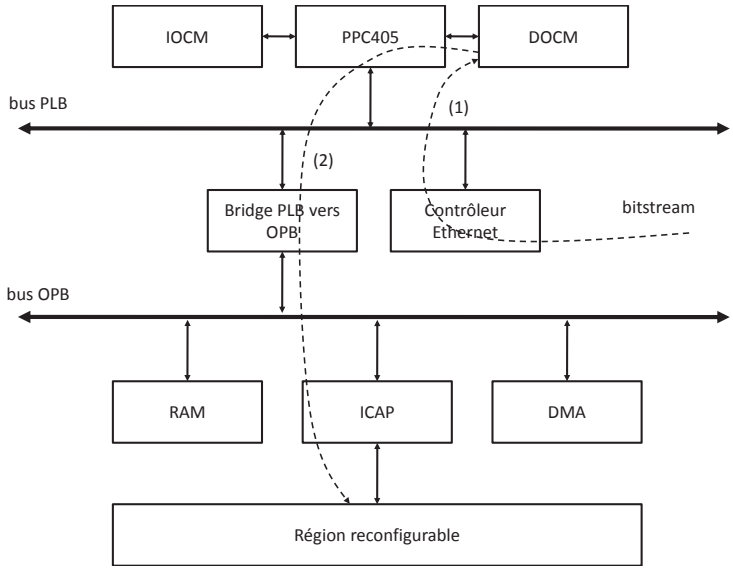

FIGURE 5.5 – Architecture matérielle du niveau L2 sans DMA

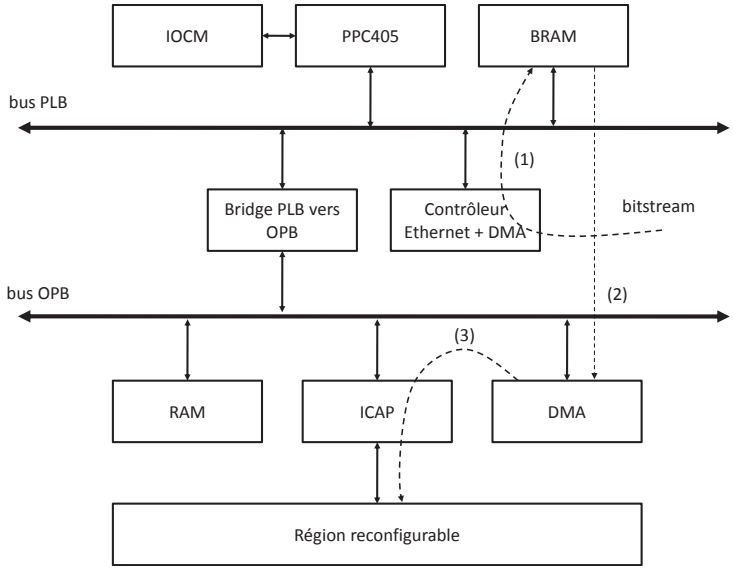

FIGURE 5.6 – Architecture matérielle du niveau L2 avec DMAs

Reed-Solomon. Ici c'est l'opposé. Plus les bits d'erreurs seront concentrés, plus le protocole sera meilleur puisqu'il rejettera tout le fichier du bitstream et donc tous les bits erronés. Grâce au cache L1 au niveau FPGA, la transmission de bitstreams est seulement nécessaire lorsqu'il n'est pas présent dans les mémoires locales et le trafic de bitstreams est réduit.

5.3.3 Architecture matérielle

La première architecture matérielle proposée en Figure 5.5 est similaire à celle de la Section 5.3 mais sans le composant DMA. La conception expose un téléchargement de bitstreams partiels à 400 Kb/(s.MHz). Les mesures montrent que le partitionnement entre le matériel et le logiciel n'est pas idéal, le goulot provenant de ce dernier. Plus de 90% du temps de calcul est passé dans le transfert de données du contrôleur Ethernet vers le tampon circulaire et du tampon circulaire vers le contrôleur de l'ICAP.

La seconde architecture matérielle proposée (Figure 5.6) est basée sur un Virtex-4 VFX 60 cadencé à 100 MHz sur une carte d'évaluation Xilinx ML410. La carte possède quatre contrôleurs Ethernet MAC 10/100/100 Mb/s embarqués. Notre architecture n'utilise qu'un seul de ces contrôleurs configurés à 100 Mb/s. Au lieu de n'utiliser que le logiciel pour le transfert de données, deux composants DMA sont utilisés dans le but de :

1. Transférer les données du contrôleur Ethernet vers le tampon circulaire.

2. Transférer les données du tampon circulaire vers l'ICAP.

Les tampons "paquets", devant être accessibles par les deux DMAs ne peuvent rester dans la mémoire DOCM du PPC qui est privée. Ces tampons sont donc stockés dans une BRAMs soit sur le bus PLB soit sur le bus OPB. Parce que les accès maîtres doivent être autorisés pour les DMAs, deux bridges de bus (PLB/OPB et OPB/PLB) doivent être ajoutés pour autoriser de tels transferts. Après tests sur Virtex-2/XUP et Virtex4/ML410, nous obtenons des résultats similaires, soit un téléchargement de bitstreams à 800 Kb/(s.MHz).

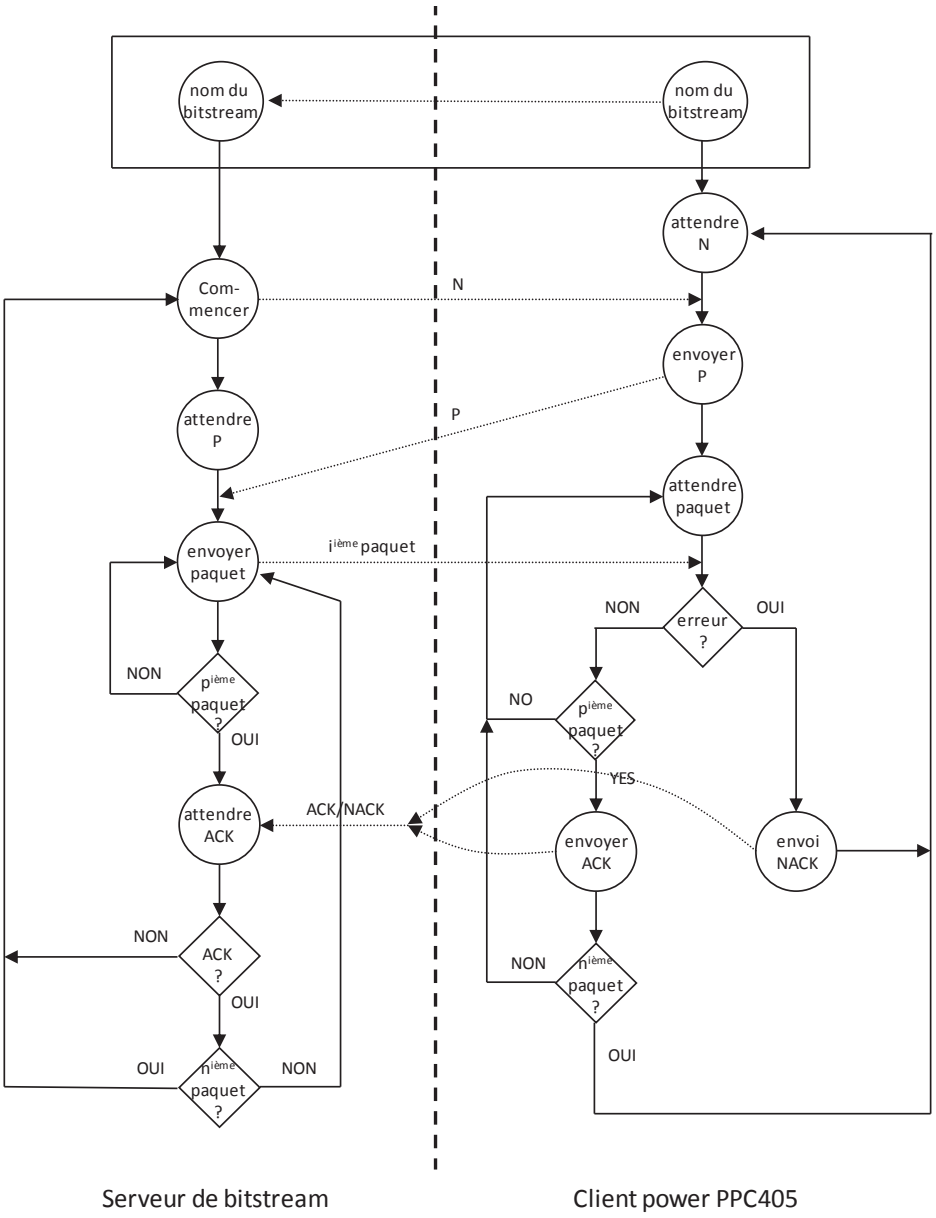

FIGURE 5.7 – Partie logicielle du protocole de reconfiguration dynamique partielle

5.3.4 Architecture logicielle

L'architecture logicielle est basée sur trois modules : le pilote ICAP, le pilote Ethernet et le protocole RP (Figure 5.7). L'objectif est de réduire le nombre de couches logicielles à traverser lorsqu'un bitstream s'écoule du contrôleur Ethernet vers le port ICAP. Un module de mesure de temps basé sur un timer matériel interne du PPC405 ainsi que les accès au lien série via la bibliothèque libc de Xilinx, sont utilisés et ne seront pas commentés, leurs utilisations étant marginales. Ce logiciel établit un pipeline de données entre le serveur de bitstreams distant et les régions reconfigurables du FPGA. Nous pouvons planifier d'atteindre la bande passante maximum Ethernet de 100 Mb/s et aujourd'hui avec notre débit de 800 Kb/(s.MHz), nous atteignons un débit soutenu de 80 Mb/s. Pour découpler le téléchargement de l'ICAP de la réception de paquet Ethernet, nous pouvons concevoir en logiciel un paradigme producteur-consommateur : le producteur étant le contrôleur Ethernet et le consommateur étant le port ICAP. Un tampon circulaire est nourri de façon asynchrone avec les paquets du composant privé DMA du contrôleur Ethernet. La réception des paquets se fait par bursts : plusieurs paquets sont reçus sans flot de contrôle. La taille des burst de paquets (P) est plus petite ou égale à la moitié de la capacité des tampons des paquets. Chaque paquet Ethernet a une taille maximale de 1518 octets avec une charge utile de 1500 octets de données de bitstream. Le protocole RP est exécuté de manière concurrente avec les gestionnaires d'interruptions Ethernet. Il analyse le contenu des paquets et transfère le bitstream du buffer vers le port ICAP via un second composant DMA. Le dimensionnement du tampon intermédiaire est un point critique en termes de performances. Plus gros est la taille du burst, plus rapide est le protocole. La possible taille du tampon dépend de la mémoire disponible au moment de la reconfiguration et cette ressource rare peut changer au cours du temps. Le protocole adapte dynamiquement les tailles de bursts à la taille du tampon. (Bomel *et al.*, 2008) donne les détails de la spécification.

5.3.5 Résultats

Les résultats obtenus (Figure 5.8) dépendent aussi, comme nous pouvons le prévoir, de la taille des tampons paquets producteur-consommateur alloués au protocole de RP. Les courbes du dessus représentent respectivement de bas en haut, les vitesses mesurées pour la première architecture et pour la

seconde. Dans tous les cas, lorsque la taille des bursts est plus grande ou égale à trois paquets (P=3), des vitesses maximales de 400 Mb/(s.MHz) et de 800 Mb/(s.MHz) sont atteintes et sont stables. La taille du tampon circulaire étant de taille 2P, de la place pour exactement six paquets est nécessaire soit 9 KB (6× 1.5 Ko) seulement. Comparativement aux tampons de plusieurs centaines de Ko des piles de protocoles standards, cette grandeur est faible pour un service continu de RP.

Les lignes plates du bas, représentent les vitesses moyennes obtenues par Xilinx, Lagger et probablement Williams. Notre protocole de RP montre une vitesse de reconfiguration de 80 Mb/s, proche de notre limite locale de 100 Mb/s. Le fossé entre les vitesses de reconfiguration et la vitesse de l'ICAP est maintenant d'un seul ordre de magnitude au lieu de trois ordres de magnitude. Enfin, notre logiciel de RP tient dans 32 Ko de mémoires données et 40 Ko de code exécutable. Comparé aux autres travaux, l'endo-reconfiguration que nous atteignons avec notre approche est 20 fois plus efficace et demande un espace mémoire moins important.

5.4 Niveau de hiérarchie L3

Un lien ad-hoc spécifique de données est utile lorsqu'il n'y a pas de routage IP et lorsqu'un peu de ressources matérielles et logicielles sont disponibles. Cependant, il est nécessaire d'être capable de télécharger un bitstream présent sur n'importe quelle machine. L'utilisation d'un standard réseau TCP/IP est parfaitement compatible lorsqu'une configuration distante est voulue. Le niveau L3 est le niveau réseau local large (Wide Area Network, WAN) où la latence est d'environ 100 ms puisque sa position géographique est la plus lointaine.

5.4.1 Protocoles de transport communément employés

(Rind *et al.*, 2006) guide dans les choix envers TCP (Transport Communication Protocol) contre UDP (User Datagram Protocol) et vice-versa par des métriques comme la rapidité, le nombre de dispositifs mobiles et la capacité du lien, donc sa bande passante. Les résultats sont donnés en terme de débit par un simulateur de réseaux. Il montre que TCP donne de meilleures performances lorsqu'un minimum de dispositif est connecté à un réseau lo-

cal sand-fil (Wireless LAN, WLAN) et montre clairement que les noeuds qui se déplacent perturbent la transmission de paquets. UDP est trouvé meilleur quand il est possible d'accepter une petite perte de paquets, c'est donc un bon choix lorsque l'on souhaite privilégier une livraison rapide de données. Uchida (Uchida, 2007) présente un processeur TCP matériel pour la norme Ethernet Gigabit qui ne requiert qu'un seul dispositif physique Ethernet PHY. Le circuit est suffisamment petit pour être implémenté sur un seul FPGA, avec un débit annoncé de 949 Mb/s en émission et en réception. Avec cette approche, aucun traffic important sur le réseau est considéré et le contrôle de congestion est bien connu pour être conçu et optimisé pour les réseaux filaires. La norme International IEEE 802.11 (WIFI) décrit les caractéristiques liées aux LAN sans-fil. Elle rend possible la construction de réseaux sans-fil larges bandes et donne accès en pratique à une connection entre les ordinateurs, ordinateurs portables, assistant numérique personnel (Personal Digital Assistant, PDA), dispositif communiquant et autre périphérique, de façon intérieure ou extérieure avec des portées, rapidité et modes dépendants des révisions nombreuses du standard allant de 802.11a à 802.11s. Le sans-fil est parfois la seule possibilité pour les applications qui demandent une grande mobilité. En industrie, la réduction des câblages est pertinente puisqu'elle impose des limitations strictes en coût d'installation de réparation et de placement. Comparé au Bluetooth, la principale force du Wi-Fi se tient dans son haut-débit et sa grande portée. Comme le système cible utilisera un lien Wi-Fi et est limité à un débit bien inférieur, les taux de transfert type de l'ordre du gigabit sont sur-dimensionnés. Ploplys dans (Ploplys et Alleyne, 2003) a effectué une étude où un protocole "sans fil" UDP est utilisé pour des performances temps-réel, dans le cadre d'application de contrôle. La perte de données est bien définie, expliquée et évaluée et basée sur des facteurs comme la portée, les obstacles environnementaux, les charges de calculs et l'augmentation du traffic réseau. Les travaux existants montrent que TCP est vastement employé dans les topologies LAN et UDP pour les WLAN. L'utilisation d'UDP est donc naturelle si l'on cible des dispositifs portables sans-fil.

5.4.2 Modèle d'architecture TCP/IP

TCP/IP (RFC1122, 1989) est une suite de références réseaux utilisées pour développer des applications réseaux. Ce modèle est fréquemment décrit comme possédant 4 couches, comme montré en Figure 5.9. Pour une

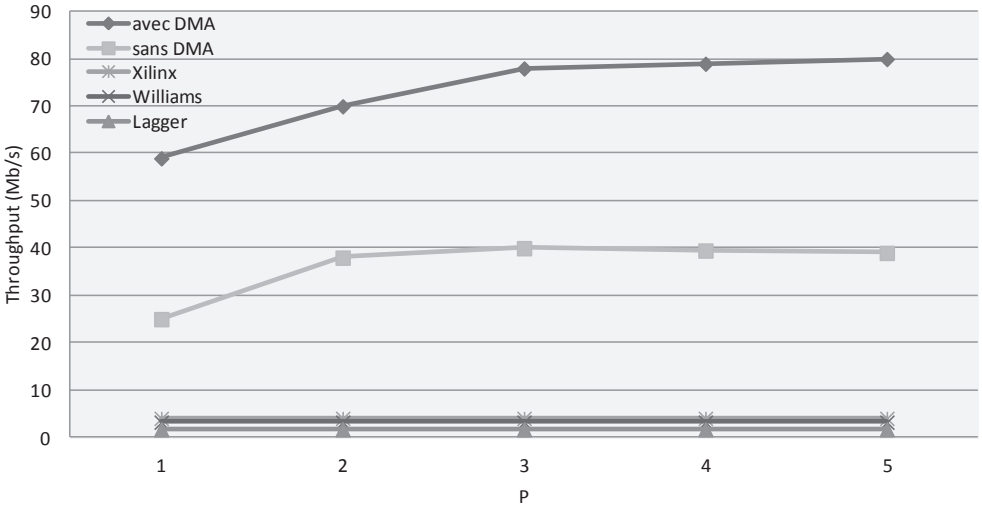

FIGURE 5.8 – Débit de l'endo-reconfiguration en fonction de la taille de burst P

transmission, les données sont envoyées de la couche 4 vers la couche 1, et inversement pour une réception. Dans les lignes suivantes, nous plaçons la discussion aux niveaux 1 et 3.

Aujourd'hui, les protocoles IP sont adaptés pour les transferts de données faible latence et hauts débits. Il faut donc choisir d'adapter notre protocole précédent avec l'un des plus fréquemment utilisés pour le transport (couche 3). TCP (RFC793, 1981) est utilisé pour les applications non-critiques comme HTTP, FTP et SMTP. Il est orienté connexion et garantit que le récepteur recevra exactement ce que l'émetteur envoie, sans erreur et dans le bon ordre. Pour éviter la congestion, TCP diminue la vitesse lorsqu'un paquet est perdu et la réaugmente lentement lorsque les paquets sont correctement transmis. Comme pour la plupart des protocoles de communication, nous notons que les pertes de paquets pour TCP sont attribuées à la congestion, i.e. un traffic dense. Alors, pour un environnement sans-fil le taux erreur de transmission de bits est élevé, les performances de TCP sont largement dégradées par le mécanisme de contrôle de congestion. UDP (RFC768, 1980) est similaire à TCP et se trouve dans la même couche TCP/IP avec pour applications les plus courantes le système de noms de domaine (Domain Name System, DNS) et le protocole simple de gestion de réseau (Simple Network Management Protocol, SNMP). Il est orienté non-connexion et différencié par la relation entre l'émetteur et le récepteur. L'émetteur peut envoyer des données à un récepteur, mais UDP ne fournit pas de système de fiabilité de réception. Il n'y a donc aucune garantie qu'un paquet arrivera sans corruption. UDP est plus rapide que TCP puisqu'il n'y a pas de pénalité supplémentaire pour la vérification d'erreur. Une comparaison entre TCP et UDP est donnée par le Tableau 5.2 avec comme référence (Instruments, 2009). UDP est donc trouvé comme étant le protocole le plus acceptable pour la diffusion de bitstreams sans-fil. Et si erreurs de transmission il y a, la totalité des paquets serait retransmise.

5.4.3 Architecture logicielle

Nous différencions le logiciel s'éxecutant sur le serveur et le logiciel exécuté par le processeur du FPGA.

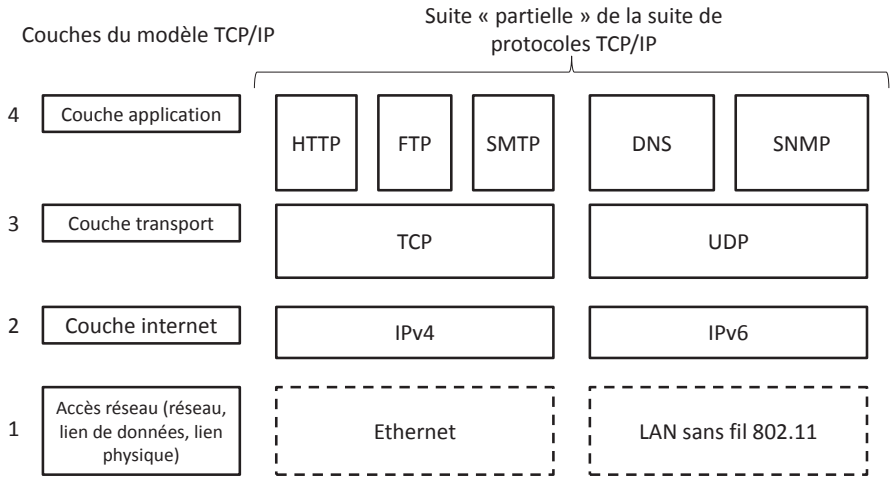

FIGURE 5.9 – Les cinq couches du modèle de l'architecture TCP/IP

5.4.3.1 lwIP comme pile réseau TCP/IP

Au lieu de développer une bibliothèque réseau complète, nous avons choisi la pile TCP/IP open source pour les systèmes embarqués lwIP. Directement disponible pour EDK, lwIP (Dunkels, 2001) est une implémentation sous licence de Berkeley (Berkeley Software Distribution, BSD) d'une pile TCP/IP avec une utilisation mémoire très faible. Elle a été initialement développée par Adam Dunkels de la l'Institut Suédoise d'Informatique (Swedish Institute of Computer Science) et est désormais maintenue par quelques développeurs dirigés par Leon Woestendberg et est hébergée par Savannah. Le portage proposé par Xilinx pour EDK est robuste et possède un grand nombre de paramètres configurables à la compilation permettant de personnaliser la pile selon les nécessités des applications. lwIP peut également être utilisé avec un système d'exploitation ou non.

Notre première approche a été de "tailler" la bibliothèque lwIP pour n'utiliser qu'UDP uniquement puisqu'aucun autre protocole n'est nécessaire. Pour assurer le paradigme producteur-consommateur, la pile lwIP utilise une mémoire de tampons. Cette mémoire est un point critique en termes de performances et doit être bien dimensionnée. Les valeurs par défaut de ce segment sont proches des 800 Ko soit 512 paquets de 1528 octets. Les tests émission-transmission continue de paquets montrent qu'une allocation de 100 Ko pour la mémoire tampon est pertinente sans interférer sur les performances. Par la suite, en déchargeant le processeur de calculs lourds comme les vérifications des sommes de contrôle des paquets UDP, nous obtenons un gain en débit multiplié par 2. Ce calcul de sommes est effectué par le contrôleur Ethernet à l'émission et à la réception.

5.4.3.2 Protocole logiciel de RDP

Nous décrivons maintenant notre protocole en introduisant la structure de la trame retenue. Elle est constituée de deux champs, le premier étant réservé pour décrire la taille de la trame (10 octets) et le second étant la charge utile, i.e. le bitstream. La taille maximale des paquets est liée au format de trame que nous utilisons (DIX Ethernet Version II). La taille maximale d'un paquet Ethernet est 1528 octets. Il inclut 20 octets de préambule, 4 octets de vérification de séquences (Frame Check Sequence, FCS), 6 octets pour l'adresse MAC de destination, 6 octets pour l'adresse MAC de la source et 2

Protocole	Complexité	Vitesse	Architecture	Fiable
UDP	Faible	Haute	Diffusion Client/Serveur	Non
TCP	Moyenne	Faible	Client/Serveur	Oui

Tableau 5.2 – Comparaison des protocoles TCP et UDP

octets pour le type de la trame. De ces 1500 octets restants, 20 octets sont utilisés pour l'en-tête IP et 8 pour l'en-tête UDP. Un total de 1472 octets de données est donc disponible pour le bitstream pour un paquet. Un bitstream étant généralement plus grand que cette taille, il sera fragmenté avant l'émission. La transmission de paquets est synchronisée par une poignée de mains pour assurer la transmission correcte des données. Le protocole peut assurer deux modes de fonctionnement, esclave et maître, comme montré respectivement par les Figures 5.10 et 5.11. En mode maître, l'architecture au sein du FPGA demande elle-même au serveur LAN un bitstream partiel. En mode esclave, c'est le serveur qui force la mise à jour d'une région du FPGA. Évidemment, obtenir le meilleur débit possible demande des précautions d'usage qui concernent l'écriture d'un bitstream partiel dans la région de reconfiguration. D'une perte de paquets résultera une réception incomplète du bitstream et donc l'impossibilité non seulement de reconfigurer le FPGA mais pourrait également mener à des comportements non souhaitables et indéterminés. Pour éviter cela, un identifiant de trame est utilisé pour valider la mauvaise ou bonne réception (perte et ordre). Avant la commande de l'ICAP pour l'écriture dans la région reconfigurable, un contrôle de redondance cyclique (Cyclic Redundancy Check, CRC) est également effectué.

5.4.4 Architecture matérielle

L'architecture matérielle est similaire à celle de la Section 5.3. Cependant, le protocole de reconfiguration dynamique partielle doit être détaillé en profondeur. Une réception de paquets est scindée en quatre étapes, décrit par la Figure 5.12, et va de la pile premier arrivé premier servi (First in first out, FIFO) interne au contrôleur Ethernet jusqu'à la région reconfigurable :

1. Premièrement, une interruption en réception intervient. Le paquet est stocké dans une FIFO interne au contrôleur Ethernet.

2. Deuxièmement, le tampon FIFO est copié vers la RAM où le tampon circulaire alloué par le gestionnaire mémoire de lwIP est disponible.

3. Troisièmement, le contenu du tampon circulaire est transféré vers la mémoire BRAM de l'ICAP.

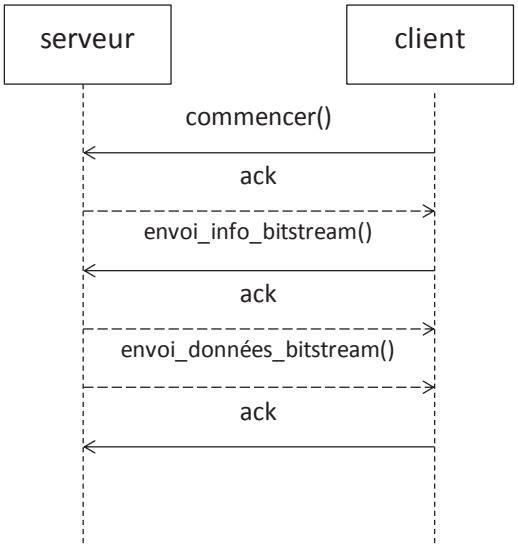

FIGURE 5.10 – Diagramme séquence du mode maître

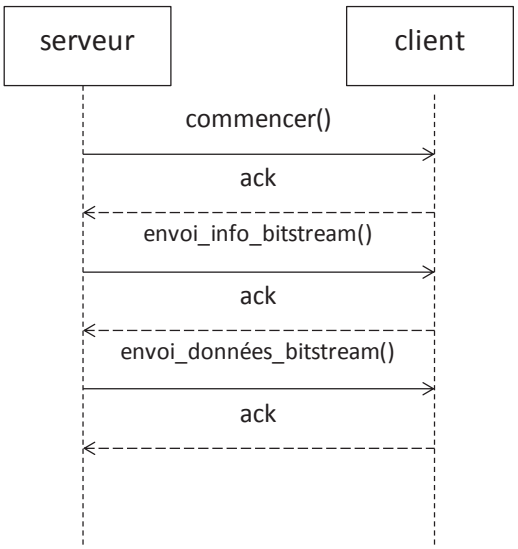

FIGURE 5.11 – Diagramme séquence du mode esclave

4. Enfin, le contenu de la mémoire BRAM de l'ICAP est écrit dans la région reconfigurable.

Pour assurer le meilleur partitionnement logiciel/matériel, nous proposons une étude autour de quatre variations architecturales élémentaires :

1. Cache de données du processeur activé.

2. Cache de données du processeur désactivé.

3. Cache de données du processeur activé et un composant DMA pour le transfert du tampon circulaire vers la mémoire BRAM de l'ICAP.

4. Cache de données du processeur désactivé et un composant DMA pour le transfert du tampon circulaire vers la mémoire BRAM de l'ICAP.

Dans tous les cas, le cache instruction du PPC405 est activé et fixé à 16 Ko (8 BRAMs). Lorsqu'il est activé, le cache de données est également de cette taille, soit 16 Ko. Nos évaluations montrent très clairement que la copie logicielle avec le cache de données activé est la meilleure configuration en termes de performances. Ceci est explicable puisque le composant DMA ne possède pas de système de mise en mémoire tampon, et n'effectue pas de transfert burst. Pour un processeur sans cache instruction, l'addition du composant DMA pourrait avoir du sens, sinon la boucle interne de copie mémoire est mise en cache et est exécutée avec 2 cycles par instruction. Le facteur limitant deviendrait la latence du bus OPB (lecture/écriture de/vers la mémoire RAM). Bien sûr, lorsqu'un cache de données est actif, il provoque des incohérences de cache et doit rester "propre". C'est de la responsabilité du développeur que d'assurer que les tampons en cache qui sont passés au DMA sont évincés. De plus le cache doit être invalidé avant l'utilisation de tampons résultats d'une opération DMA. C'est pour cela que nous avons décidé de n'utiliser que le logiciel concernant les transferts de données. De plus, nous conservons ainsi des ressources matérielles et diminuons significativement les pénalités dûes aux bridges OPB et PLB. Le PPC est également relâché de contraintes de management.

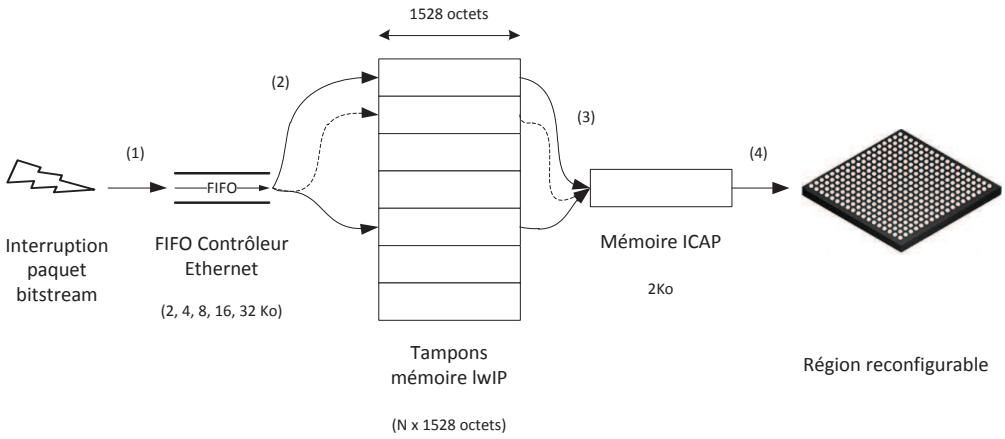

FIGURE 5.12 – Chemin des données du bitstream de l'interruption jusqu'à l'ICAP

5.4.5 Résultats

Il est visible que la taille optimale des paquets donnant le meilleur débit est celle de l'unité de transmission Ethernet maximale (Maximum Ethernet Transmission Unit, MTU). La taille du burst est également à son maximum soit une taille similaire au nombre de trames d'un bitstream. La FIFO du contrôleur Ethernet est fixé à 8 Ko. Le serveur exécute le protocole de RP en mode esclave. La Figure 5.15 résume les résultats de débits en fonction de la taille des paquets et dépendamment des configurations réseaux données en Figure 5.13 et Figure 5.14.

Pour évaluer l'efficacité générale, la configuration LAN est d'abord envisagée. Cette configuration lie directement les machines clients et serveurs via un cable Ethernet croisé. Les résultats obtenus dépendent comme nous pouvons le prévoir de la taille des paquets transmis. Nous obtenons un débit soutenu de 60 Mb/s avec une taille moyenne des paquets de 1472 octets. Ce haut-débit est compatible avec le débit Wi-Fi qui permet d'atteindre 60 Mb/s.

La configuration WLAN est similaire mis à part le lien, qui est sans-fil. Le FPGA est connecté à un bridge Ethernet/Wi-Fi ce qui évite l'utilisation de pilotes spécifiques. Le type de réseau Wi-Fi est ajusté comme ad-hoc ce qui permet à deux ou plusieurs clients "sans-fil" de se connecter sans contrôleur central. Dans ce contexte un débit de 30 Mb/s est obtenu. C'est le maximum physiquement atteignable puisque les 54 Mb/s possibles par le standard 802.11g ne sont que théorie. A notre connaissance, aucun travaux ne propose une reconfiguration dynamique partielle utilisant un tel lien.

En termes de coût mémoire logiciel, 100 Ko sont dédiés au code exécutable et 100 Ko alloués pour les données. Ces nombres sont 4 fois inférieurs comparés aux travaux précédents. En terme de ressources matérielles, toutes les instructions et données sont stockées en mémoire sur-puce BRAMs et un contrôleur supplémentaire DDR RAM n'est pas nécessaire. Seulement 8% de la puce est utilisée, c'est à dire 2 524 slices sur 32 383 disponibles. Pour un Virtex-2 pro, une slice consiste en deux tables de scrutation 4 entrées et deux registres. Notre implémentation est donc légère et cette taille est adéquate pour implémenter ce circuit sur un FPGA faible-coût.

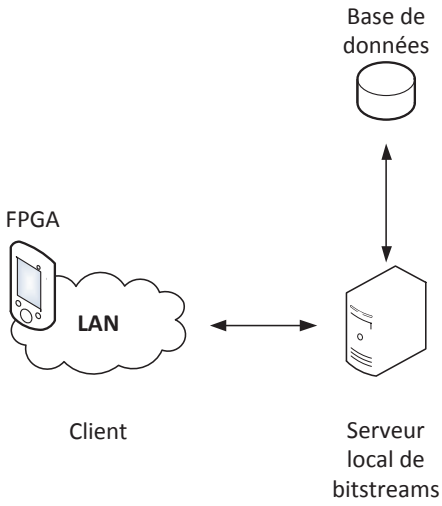

FIGURE 5.13 – Architecture de la configuration LAN

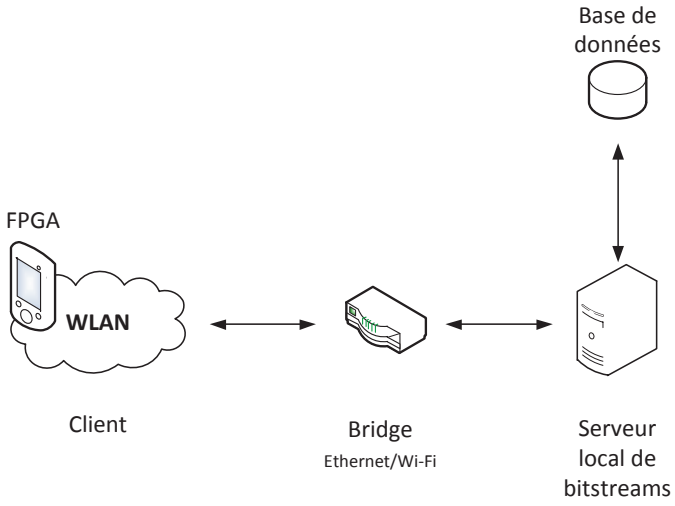

FIGURE 5.14 – Architecture de la configuration WLAN

5.4.6 Application au changement de clé GHASH

Le fait de pouvoir remplacer une région logique quelconque rend la reconfiguration dynamique partielle intéressante pour tout applicatif. L'opportunité de notre hiérarchie de dépôts peut donc être mise en perspective en l'appliquant au changement de clé de notre circuit GHASH décrit dans le chapitre 3. Ceci revient typiquement à changer complètement le module. Le problème avec cette approche est 1) le temps du re-chargement du FPGA lors du changement de clé et 2) le pré-calcul inhérent des bitstreams partiels.

Le premier point peut être partiellement adressé comme le montre le Tableau 5.3. Il dresse les différents délais de reconfiguration du module basic GHASH vu au chapitre 3 pour les différentes architectures développées. Les tailles des bitstreams partiels reportées sont liées aux bornes supérieures et inférieures des circuits GHASH. Les délais de reconfiguration constatés s'étirent de 52.0 à 7.9 ms et de 21 à 3.2 ms pour une famille de FPGA de type Virtex-5 et de 63.0 à 9.4 ms et de 20.0 à 3.0 ms pour une famille Virtex-6. Certainement pas de l'ordre du cycle, ces délais montrent qu'effectuer un changement de clé est possible dans des conditions globalement acceptables pour des systèmes temps réels.

Pour le second point, l'utilisation d'outils de synthèse et de placement-routage, quel qu'il soit est un effet indésirable dans la pratique de cette approche pour l'embarqué. Un ordinateur suffisamment puissant et équipé de licences d'outils est également indispensable limitant fortement un large déploiement.

Idéalement, un système pourrait embarquer un outil de synthèse pour produire des bitstreams partiels à-la-volée. Une telle construction reste cependant de l'ordre du fantasme au vu de la puissance à déployer pour générer de simples bitstreams. Additionnellement, les outils de synthèse se démocratisant, les concepteurs pourraient voir apparaître des FPGAs open-source leur permettant une compréhension complète de la chaîne de production. Par voie de conséquence, ceci ouvrirait le chemin à de possibles optimisations fines (méta-heuristiques pour le routage, format du bitstream, etc...) chose impossible avec les outils propriétaires. Cependant, les efforts de fonderie et l'écriture d'outils de synthèses, qui est similaire à l'écriture d'un langage de compilation haut-niveau, donnent à cet exercice un aspect complexe et blo-

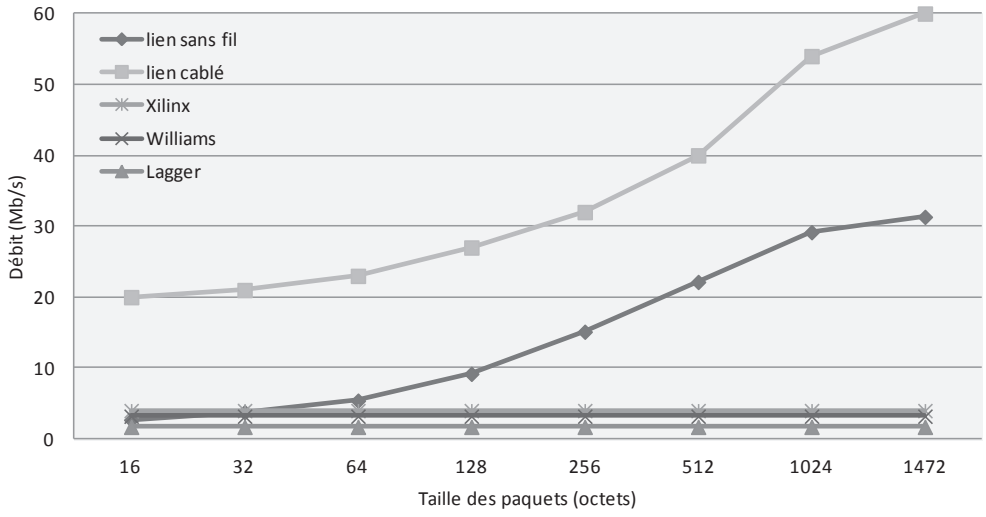

FIGURE 5.15 – Courbes des débits des configurations LAN et WLAN

quant. La majorité des développeurs sur FPGA se concentrant bien plus sur la conception matérielle que sur l'écriture d'outils. Pour voir apparaître une chaîne convenable pour FPGA, il faudrait convaincre un nombre suffisant de personnes avec des compétences logicielles et matérielles pour effectuer un effort dans ce sens.

5.5 Conclusion

La hiérarchie de dépôts proposée montre qu'il y a encore des opportunités pour améliorer le niveau cache, LAN et IP, stratégies de cache et protocoles dans le but de fournir un service de reconfiguration distant et efficace par un réseau standard. Jamais véritablement prise en considération des travaux précédents, notre plate-forme de RP prend avantage de trois niveaux spécifiques L1, L2 et L3. Pour le niveau L1, le débit de reconfiguration est de 200 Mb/s. Pour le niveau L2, le débit de reconfiguration est de 80 Mb/s soit un facteur multiplicateur de 20 comparé aux autres travaux. Pour le niveau L3, la technologie sans fil est utilisée avec un débit de 30 Mb/s ce qui est la limite physique possible. De plus, lorsqu'utilisé en topologie LAN, l'implémentation

Spartan-3 et reconfiguration partielle : Dans le chapitre précédent, nous mentionnons des cibles telles que les FPGA Spartan-3 (ou Spartan-6) comme candidats potentiels pour l'authentification haut-débit dans les systèmes embarqués. Bien que le fabless Xilinx ne les supporte pourtant pas pour la reconfiguration dynamique partielle des publications prouvent qu'il est possible de faire de la sorte (Bayar et Yurdakul (Koch *et al.*, 2010)).

Cible dépréciée Virtex-2 pro : Le FPGA Virtex-2 pro bien que historique pour ces capacités de configuration dynamique partielle, est logiquement déprécié au profit de FPGAs plus récents. Le lecteur pourra remarquer que la cible FPGA ici choisie semble être une "rétrogradation" par rapport aux chapitres précédents. L'explication se trouve très simplement dans le fait que les travaux issus de ce chapitre ont été effectués en aval. Les auteurs admettent cependant l'hypothèse que les débits des différentes architectures de ce présent chapitre seraient au moins similaire sur une plate-forme technologiquement supérieure.

	Taille bitstream partiel (Mbit)	Temps reconfiguration (ms)				
		30 Mbit/s	60 Mbit/s	40 Mbit/s	80 Mbit/s	200 Mbit/s
Virtex-5 (xc5vlx50t-3ff1136)						
Borne Supérieure	1.58	52.0	26.0	38.0	19.0	7.9
Borne Inférieure	0.64	21.0	10.5	16.0	8.0	3.2
Virtex-6 (xc6vlx75t-3ff484)						
Borne Supérieure	1.89	63.0	31.5	46.0	23.0	9.4
Borne Inférieure	0.61	20.0	10.0	15.2	7.6	3.0

Tableau 5.3 – Tailles des configurations partielles et délais de reconfiguration pour un module GHASH basic

	Lagger	Williams	Xilinx	L1 Arch1	L2 Arch1	L2 Arch2	L3 LAN	L3 WLAN
Débit (Mb/s)	1.7	3.2	4	200	40	80	60	30
Mémoire (octets)	> 1M	> 1M	< 100K	< 100K	< 100K	< 100K	> 200K	> 200K

Tableau 5.4 – Comparaison des débits et des empreintes mémoires des architectures

montre un ordre de magnitude de gain ($\times 15$) comparativement aux travaux précédents, soit 60 Mb/s. Le protocole sous-jacent est également simple et peut être réutilisé facilement avec un coût mémoire logicielle de 200 KB et seulement 8% d'un FPGA Virtex-2 pro. Le Tableau 5.4 résume les vitesses respectives et les empreintes mémoires pour chaque architecture présentée dans ce chapitre.

Chapitre 6

Un Stockage Efficace du Matériel Cryptographique

L'avènement du haut-débit permet d'explorer une nouvelle façon de consommer de l'information et les commodités pour l'utilisateur final sont effectivement nombreuses. Pourtant, nous avons précédemment évoqué l'obligation de maintenir un grand stock de données pour entretenir une sécurité accrue. Clés, valeurs d'horodatage, vecteurs d'initialisation ou tags, participent à l'explosion de la mémoire disponible sur-puce qui ne peut être prise à la légère.

Parfois passé sous silence, nous proposons d'adresser ce problème de stockage en évaluant une solution basée sur une structure probabiliste. Pour la première fois, nous proposons une approche compatible avec les systèmes embarqués. A la suite de l'introduction, la partie deux introduit et développe cette structure redécouverte il y a peu. Résultats à l'appui, la troisième partie s'intéresse à l'élaboration d'une architecture compatible pour les systèmes embarqués. La quatrième partie est une discussion autour des avantages et des inconvénients qui apporte force à la conclusion du chapitre.

Sommaire

6.1 Problème du matériel cryptographique

Avec le déploiement du haut-débit, les capacités de stockage n'ont eu cesse d'augmenter et les centres de traitements de données en sont le parfait exemple. Cette tendance illustre toute la difficulté à contrôler de tels flux. Si des débits de plusieurs Gbit/s sont atteignables, les données à stocker peuvent également être de cet ordre pour certaines applications. Et la cryptographie ne déroge malheureusement pas à la règle.

Nous traitons spécifiquement dans ce chapitre, de l'utilisation des filtres de Bloom pour le stockage du matériel cryptographique. Si l'on suit le schéma du mode GCM décrit dans le chapitre 2 et implémenté dans les chapitres 3 et 4, ce matériel se ramifie comme suit :

- **Les clés de chiffrement** : Les clés de chiffrement sont sensibles à la divulgation et à la falsification et doivent être compartimentées. Elles sont généralement stockées sur-puce dans des mémoire sécurisées. Sauf pour des infrastructures larges, le nombre de clés est limité dans le cas des systèmes embarqués.

- **Les TS et IVs** : En mode compteur, les blocs de chiffrement ont nécessairement besoin d'une valeur d'entrée qui rend aléatoire les transformations internes de l'algorithme.

- **Les valeurs AC** : Les condensats cryptographiques sont nécessaires pour l'authentification de données. Pour des messages longs, le stockage peut être limité. C'est bien moins le cas pour des messages courts, où un rapport d'un tag pour un mot peut être constaté.

Si les méthodes de compressions habituelles sont envisageables, elles requirent le plus souvent des mécanismes complexes qui augmentent considérablement les latences de calcul et la surface nécessaire.

"Où que soit utilisé un ensemble et que l'espace est une considération, un filtre de Bloom doit être envisagé. Lorsqu'un filtre de Bloom est utilisé, il est primordial d'évaluer les effets potentiels des faux positifs". Partant de cette citation (Bloom, 1970), nous avons envisagé l'utilisation de cette structure pour le stockage, non pas du matériel cryptographique dans son

ensemble, mais pour les valeurs d'authentification AC uniquement. En effet, les spécifications du filtre de Bloom nous limite à cet usage. Comme montré en Figure 6.1, le filtre prend la place de la mémoire AC et s'insère dans notre coeur de sécurité matériel vu au Chapitre 4. Bien que spécifié il y a plus de 40 ans, le filtre de Bloom est une solution de compression potentiellement robuste, qui demande à être expertisée pour une utilisation envisageable pour les systèmes embarqués.

6.2 Les filtres de Bloom

Le filtre de Bloom est une structure de données probabiliste compacte qui permet d'effectuer des requêtes d'appartenance d'un élément à un ensemble. Cette structure conçue par Burton H. Bloom en 1970 (Bloom, 1970) offre une façon efficace de représenter un ensemble pouvant provoquer des faux positifs (un élément peut être détecté comme faisant partie de l'ensemble alors qu'il ne l'est pas), mais jamais des faux négatifs (un élément détecté comme ne faisant pas partie de l'ensemble ne l'est pas). Les opérations de bases sont l'addition d'éléments et la requête d'adhésion d'un élément dans une représentation probabiliste d'un ensemble. Le lecteur trouvera dans (Tarkoma *et al.*, Second issue 2012) une étude exhaustive de cette structure et ses applications potentielles.

Le filtre de Bloom classique ne permet pas la suppression d'élément. La précision d'un filtre de Bloom dépend de la taille du filtre, du nombre de fonctions de hachage utilisé par le filtre, et du nombre d'éléments ajouté à l'ensemble. Plus le nombre d'éléments ajoutés est élevé, plus l'opération de requête pourra rapporter des faux positifs.

Un filtre de Bloom est un tableau de m bits qui représente un ensemble $S = \{x_1, x_2, ..., x_n\}$ de n éléments. Tous les bits de l'ensemble du filtre sont initialement mis à zéro et l'on utilise k fonctions de hachage $h_i(x)$ avec $1 \leq i \leq k$ pour mapper les éléments $x \in S$ aux nombres aléatoires uniformes 1,...m. Un choix classique de fonction de hachage est la fonction MD5.

L'insertion d'un élément $x \in S$ est effectuée par la mise des bits $h_i(x)$ à un pour $1 \leq i \leq k$. y est alors considéré comme membre de l'ensemble S si tous les bits $h_i(y)$ sont à un. Inversement, il n'est pas considéré comme

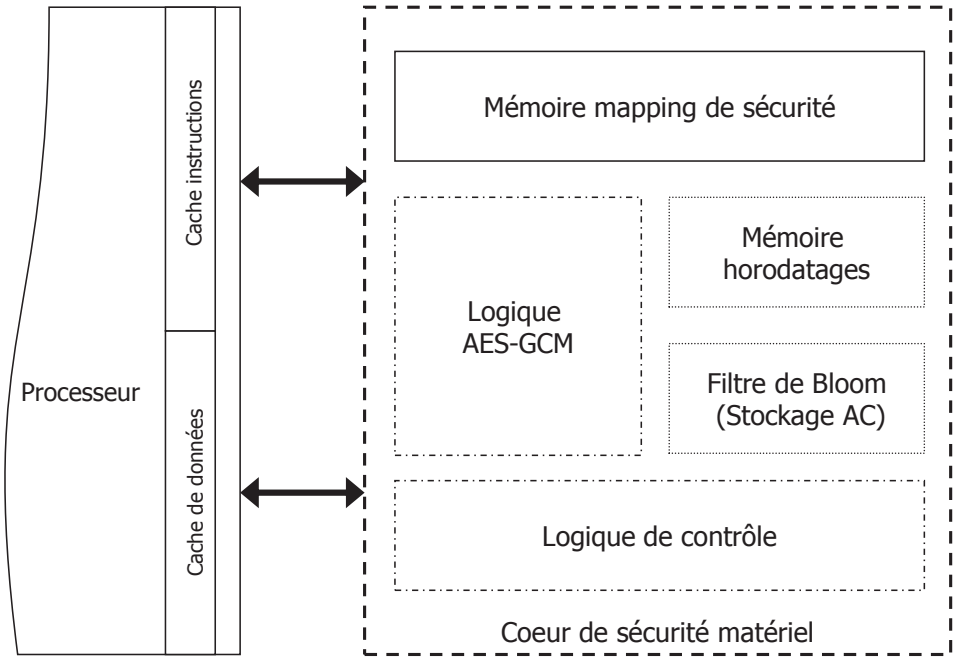

FIGURE 6.1 – Aperçu de la protection de la mémoire principale avec Filtre de Bloom

membre si un seul des bit $h_i(y)$ n'est pas un. Le point faible des filtres de Bloom se situe dans la possibilité de faux positifs. Pour rappel, les faux positifs sont des éléments ne faisant pas partie de S mais sont pourtant rapportés comme appartenant à l'ensemble du filtre. A titre d'exemple, l'insertion de trois éléments x, y et $z \in S$ est montré par la Figure 6.2. Des valeurs de hachage de ces fonctions, produites par deux fonctions de hachage h_1 et h_2, résultent les indices du tableau qui sont positionnés à un. La réponse à la scrutation de l'élément $w \in S$ est donc négative puisqu'un bit est positionné à un, et l'autre l'est à zéro, invalidant l'appartenance de w au filtre.

Dans le but d'obtenir les meilleures performances possibles, chaque fonction de hachage k doit être membre de la famille des fonctions de hachage universelles. Cela signifie que toutes ces fonctions mappent chaque élément de l'univers à un nombre aléatoire uniforme. Une solution quasi-idéale de hachage uniforme est présentée dans (Ostlin et Pagh, 2003). En pratique, les fonctions de hachage donnant des sorties suffisamment uniformément distribuées comme MD5 ou CRC sont largement utilisées tout comme les 25 fonctions de hachage évaluées de façon empirique par (Henke *et al.*, 2008).

Un filtre de Bloom basé sur S demande une taille en $O(n)$ et la requête d'adhésion en un temps en $O(1)$. Les trois paramètres clés d'un filtre de Bloom, m, n et k, influent sur le comportement du filtre. Augmenter ou réduire le nombre de fonction de hachage peut diminuer la proportion de faux positif tout en augmentant les calculs nécessaires pour l'insertion et la scrutation. Ce coût est directement proportionnel au nombre de fonction de hachage. La taille de filtre peut agir sur les exigences mémoires et la proportion de faux positifs. D'un filtre large résultera un nombre inférieur de faux positifs. Enfin, la taille de l'ensemble, qui est inséré dans le filtre, détermine également la proportion de faux positifs.

6.2.1 Probabilité de faux positif

Nous décrivons ici les formules qui mènent aux proportions de faux positif d'un filtre de Bloom et au nombre optimal de fonctions de hachage pour une proportion de faux positif donnée. Nous supposons qu'une fonction de hachage, choisit chaque position du tableau avec une probabilité égale. Soit m le nombre de bits dans le filtre de Bloom, alors la probabilité qu'un bit soit mis à un, par une fonction de hachage est :

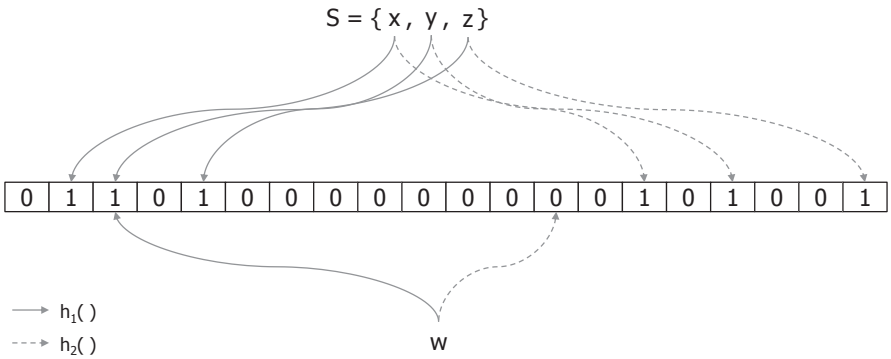

FIGURE 6.2 – Exemple d'insertion d'éléments et d'une requête à un filtre de Bloom

$$1 - \frac{1}{m} \tag{6.1}$$

Avec k fonctions, la probabilité de l'une d'entre elles de ne pas avoir mis un bit spécifique à un est donnée par :

$$(1 - \frac{1}{m})^k \tag{6.2}$$

Après avoir inséré n éléments dans le filtre, la probabilité qu'un élément soit toujours à zéro est :

$$(1 - \frac{1}{m})^k n \tag{6.3}$$

Conséquemment la probabilité que le bit soit à un est donnée par :

$$1 - (1 - \frac{1}{m})^k n \tag{6.4}$$

Pour le test d'appartenance d'un élément, si toutes les positions k du tableau du filtre calculées par les fonctions de hachage sont mises à un, le filtre de Bloom clame que l'élément appartient à l'ensemble. La probabilité que ceci arrive lorsqu'un élément ne fait pas parti de l'ensemble est :

$$(1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k \tag{6.5}$$

La proportion de faux positifs décroît lorsque la taille m du filtre de Bloom augmente. La probabilité de faux positifs augmente avec le nombre d'élément ajouté n. Pour minimiser la probabilité de faux positifs, il faut donc minimiser le terme $(1 - e^{-kn/m})^k$. On utilise la dérivée et l'égalisation à zéro, donnant le valeur optimale de k :

$$k_{opt} = \frac{m}{n} ln\ 2 \approx \frac{9m}{13n} \tag{6.6}$$

Ce résultat donne une probabilité de faux positifs de :

$$(\frac{1}{2})^k = 0.6185^{\frac{m}{n}} \tag{6.7}$$

En utilisant le nombre optimal de hachages k_{opt}, la probabilité de faux positifs peut être bornée :

$$(\frac{m}{n})^k \geq \frac{1}{ln2} \tag{6.8}$$

Pour maintenir une probabilité de faux positifs fixe, la taille du filtre de Bloom doit augmenter linéairement avec le nombre d'éléments insérés dans le filtre. Le nombre de bits m pour un nombre désiré d'éléments n et une proportion de faux positifs p, est donné par :

$$m = \frac{-n\ ln\ p}{(ln\ 2)^2} \tag{6.9}$$

Il y a un facteur $log_e \approx 1.44$ entre l'espace utilisé par le filtre de Bloom et l'espace optimal qui peut être utilisé. D'autres structures de données avec un espace plus proche de la borne inférieure existent, mais sont plus complexes (Pagh *et al.*, 2005) (Porat, 2009).

6.2.2 Autres opérations

Les filtres de Bloom standards ne supportent pas la suppression d'éléments. Cette opération peut être implémentée à l'aide d'un second filtre de

Faux positif : Récemment, (Bose *et al.*, 2008) ont montré que l'analyse originelle de la probabilité de faux positifs donnés par Bloom (et répétés dans les articles suivants) est optimiste et uniquement correcte pour des filtres de Bloom avec une taille large. L'analyse revisitée prouve que l'estimation communément employée est une borne inférieure et que la proportion de faux positifs est plus grande que prévue en théorie. C'est particulièrement le cas pour de petites valeurs de m.

Bloom qui contient les éléments ayant été retirés. Le problème de cette approche est que des faux positifs du second filtre résulte des faux négatifs dans le filtre composite ce qui est indésirable. Un nombre de structures dédiées est alors proposé dans la littérature.

Un nombre d'opérations impliquant les filtres de Bloom peut être implémenté facilement comme l'union. La nature vecteur de bits du filtre de Bloom permet l'union de deux ou plusieurs filtres de Bloom en effectuant un simple OU logique sur les vecteurs de bits. Étant donné deux ensembles S_1 et S_2, un filtre de Bloom B qui représente l'union $S = S_1 \cup S_2$ peut être construit en prenant le OU des deux filtre de Bloom $B = B_1 \vee B_2$ et en supposant que m et les fonctions de hachage sont les mêmes. Le filtre fusionné B rapportant qu'un élément appartenant à S_1 ou S_2 appartiendra à l'ensemble S.

Les filtres de Bloom peuvent être utilisés pour approximer une intersection d'ensemble. Une approche quasi-directe est d'envisager m et des fonctions de hachage similaires, et d'effectuer l'opération ET logique entre les deux vecteurs de bits.

6.2.3 Techniques de hachage

Les fonctions de hachage sont les blocs de construction clé des filtres probabilistes. Il existe une large littérature sur les fonctions de hachage depuis l'analyse aléatoire jusqu'à l'évaluation de sécurité sur les réseaux (Henke *et al.*, 2008), (Marsaglia et Tsang, 2002), (Varghese, 2004), (Kirsch *et al.*, 2010).

Une propriété importante des filtres de Bloom est la performance des faux positifs dépendant seulement du rapport bit-par-élément et pas de la forme ni de la taille des éléments hachés. Tant que la taille de ces éléments peut être bornée, le temps de hachage peut être considéré comme un facteur constant.

Nous décrivons brièvement les techniques de hachage qui sont la base pour une bonne implémentation des filtres de Bloom.

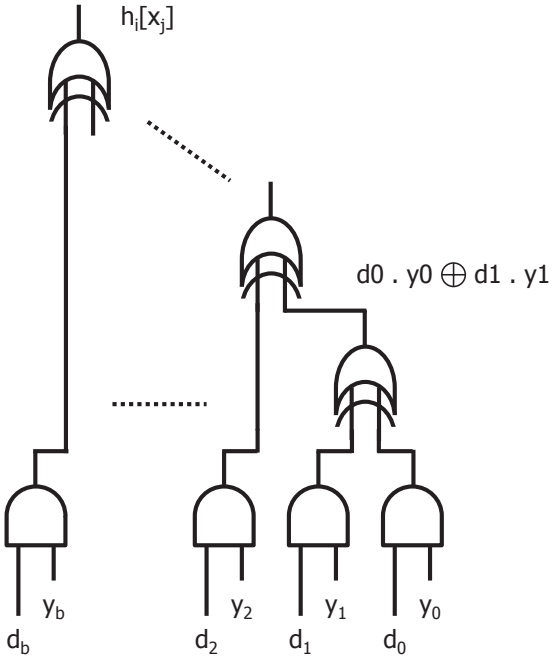

FIGURE 6.3 – Architecture de la fonction de hachage h3

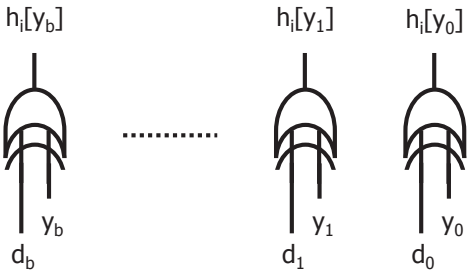

FIGURE 6.4 – Architecture de la fonction de hachage hx

6.2.3.1 Schéma de hachage parfait

Une technique simple appelée hachage parfait peut être utilisée pour stocker un ensemble de valeurs statiques S d'une façon optimale en utilisant une fonction de hachage parfaite. Une telle fonction est une injection de S dans un tableau $|S| = n$ cases de hachage. Ce tableau de taille n est utilisé pour stocker les informations associées à chaque élément $x \in S$.

Une fonctionnalité similaire au filtre de Bloom peut être obtenue en trouvant une fonction de hachage parfaite P et en stockant à chaque position le condensat de taille $f = 1/\epsilon$, calculé en utilisant une fonction H de hachage pseudo aléatoire ou aléatoire.

La scrutation de x consiste simplement à calculer $P(x)$ et à vérifier si la valeur de la fonction de hachage correspond à $H(x)$. Lorsque $x \in S$, la valeur correcte est toujours retournée, et lorsque $x \in S$ un faux positif survient avec une probabilité au plus de ϵ. Ceci suit la définition de Carter et Wegman (Carter et Wegman, 1977), qui statue qu'un élément y qui n'est pas dans S possède une probabilité, au plus ϵ, d'avoir une même valeur de hachage $h(y)$ que l'élément dans S qui correspond à la même entrée du tableau.

Bien qu'efficace en espace, cette approche est peu considérée pour les environnements dynamiques puisque la fonction de hachage parfaite nécessite d'être recalculée lorsque l'ensemble S change.

6.2.3.2 Hachage double

L'amélioration de la technique du hachage double sur le hachage classique est d'être capable de générer k valeurs de hachage basées sur seulement deux fonctions de hachage universel comme générateurs. Les filtres de Bloom peuvent alors être construits avec moins d'opérations de hachage. Kirsh et Mitzenmacher montrent dans (Kirsch et Mitzenmacher, 2008) que seuls deux fonctions de hachage indépendantes $h_1(x)$ et $h_2(x)$ sont nécessaires pour générer des hachés supplémentaires :

$$h_i(x) = h_1(x) + f(i) \times h_2(x) \tag{6.10}$$

Où i est l'index de la valeur de hachage, $f(i)$ peut être n'importe quelle

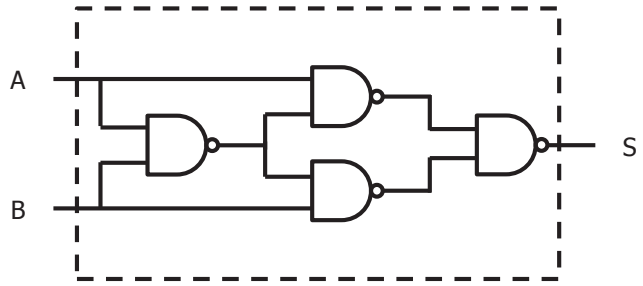

FIGURE 6.5 – Equivalence de la porte ou-exclusif à deux entrées en portes universelles non-et

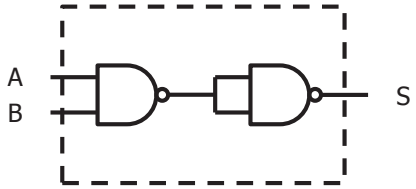

FIGURE 6.6 – Equivalence de la porte et à deux entrées en portes universelles non-et

fonction arbitraire de i et x est l'élément étant haché. Pour chaque opération du filtre de Bloom, le schéma de hachage double réduit le nombre de calculs de hachage de k à 2 sans pour autant augmenter la probabilité asymptotique de faux positifs.

6.2.3.3 Hachage par fonctions simples

Une hypothèse commune est de considérer les valeurs de hachage comme étant complètement aléatoires c'est à dire que chaque élément haché est indépendamment associé à une position uniforme. Cependant les implémentations de fonctions de hachage sont connues pour ne pas remplir cette supposition. D'un autre côté les travaux empiriques qui utilisent le hachage universel standard rapportent des différences négligeables entre les performances pratiques et les prédictions considérant un hachage idéal.

Mitzenmacher et Vadhany (Mitzenmacher et Vadhan, 2008) fournissent l'explication formelle de la différence qui subsiste entre la théorie et la pratique du hachage. Ils expliquent que ceci est dû à la combinaison aléatoire entre le choix de la fonction de hachage et l'aspect aléatoire de la donnée elle-même. Une fonction de hachage avec des propriétés aléatoires faibles est alors suffisante pour mimer en pratique une fonction aléatoire vraie. Ces résultats s'appliquent pour des techniques de hachage communes et les auteurs soulignent que les fonctions usuelles type CRC32 sont bien adaptées aux filtres de Bloom.

6.3 Une architecture pour les systèmes embarqués

Pour proposer une architecture compatible avec les systèmes embarqués, nous préconisons d'étudier les éléments ayant de fortes influences dans le chemin de données. C'est notamment le cas des fonctions de hachage qui, comme vu précédemment, forment le coeur et la robustesse de la structure. Dans cette section nous proposons de revenir tout d'abord sur le choix de ces fonctions puis par la suite, d'évoquer la structure dans son ensemble.

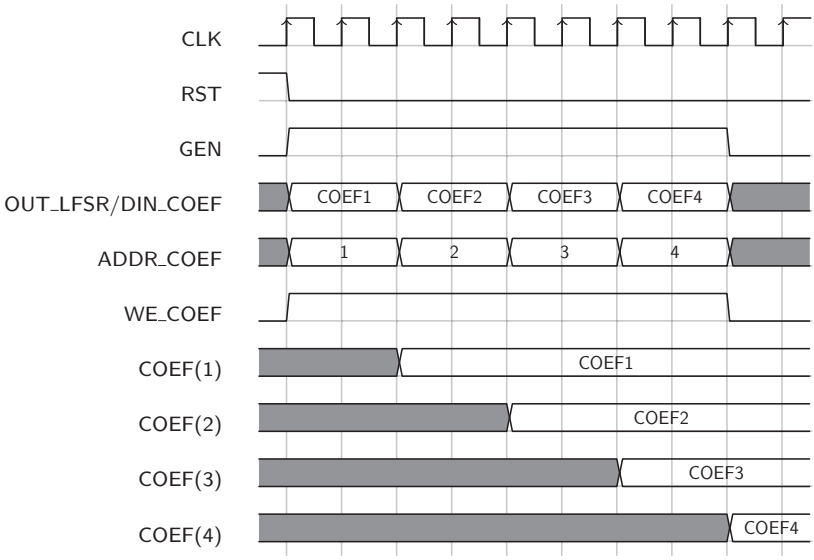

FIGURE 6.7 – Chronogrammes de génération des coefficients

6.3.1 Hachage adapté à l'implémentation matérielle

Dans le but de proposer la meilleure implémentation matérielle adaptée aux systèmes embarqués, nous proposons d'évaluer deux fonctions de hachage : une fonction de hachage universelle et une fonction simplifiée basée sur l'opération ou-exclusif. Ces deux fonctions doivent répondre à des propriétés essentielles garantissant leur robustesse à l'usage. A savoir :

- **Le bas-coût** : Le coût de la fonction doit être minimal.
- **Le déterministe** : La valeur de hachage d'un élément doit toujours être la même.
- **L'uniformité** : Chaque valeur de hachage doit être uniformément distribuée.

6.3.1.1 Construction des fonctions de hachage h3 et hx

La famille de fonctions de hachage universelles h3 (Carter et Wegman, 1977) a été largement indiquée comme étant idéale pour être implémentée en matériel, et spécialement sur FPGA (Dharmapurikar *et al.*, 2004b) (Song *et al.*, 2005) (Dharmapurikar *et al.*, 2004a).

Soit un ensemble de données d'entrées $x = x_1, x_2, x_3, ...x_n$ de taille m bits $y_j = y_1, y_2, y_3, ...y_b$, la $i^{ème}$ fonction de hachage de la famille universelle h3 de la chaîne de bits y_j est donnée par l'équation :

$$h_i(x_j) = d_{i1}.y_1 \oplus d_{i2}.y_2 \oplus ... \oplus d_{ib}.y_b \tag{6.11}$$

Où $h_i(x_j)$ est la $i^{ème}$ fonction de hachage de la $j^{ème}$ chaîne d'entrée de l'ensemble, d_{ij} est un coefficient aléatoire allant de 1 à m, les y_j sont les bits en particulier de la chaîne d'entrée, . est l'opérateur logique et, et $\oplus$ est l'opérateur ou-exclusif. L'architecture d'une telle fonction de hachage est montrée en Figure 6.3

En utilisant ces mêmes notations, le hachage par ou-exclusif hx (Figure 6.4) est quant à lui défini par la simple équation suivante :

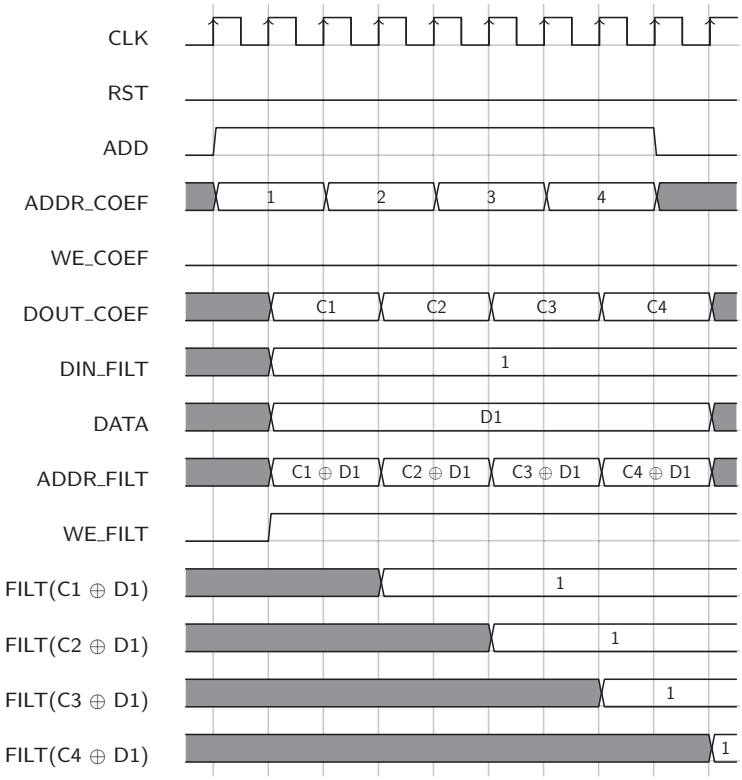

FIGURE 6.8 – Chronogrammes de la programmation du filtre

$$h_i(x_j) = d_{i1} \oplus y_1 \tag{6.12}$$

6.3.1.2 Comparaison

Pour se montrer équitable, notre évaluation est basée sur deux critères : la surface et la latence. A partir de ces deux métriques et d'une discussion sur les propriétés du hachage, nous serons en mesure d'argumenter sur la fonction la plus adaptée à l'implémentation d'un filtre de Bloom, sur FPGA.

L'équivalence en portes universelles non-et est proposée pour évaluer la surface des circuits. Nous donnons en Figure 6.5 le schéma équivalent de la porte et, en Figure 6.6 celui de la porte ou-exclusif. Les équations (6.13) et (6.14) donnent la surface en nombre de portes non-et, pour ces deux mêmes portes et les équations (6.15) et (6.16), le temps de propagation. Appliquons ces équations de surface et de délais, et considérons à titre d'exemple le hachage d'un mot arbitraire de taille 32 bits avec les fonctions h3 et hx.

Pour le hachage h3, la surface nécessaire est donnée par l'équation (6.19) et est de $(6 \times 32 - 1) = 191$ portes non-et. Un temps de propagation issu de l'équation (6.25) donne une estimation de $(3 \times 32 - 1) = 95$ portes non-et. Pour le hachage hx, l'équation (6.22) donne une surface de $4 \times 32 = 128$ portes non-et, et l'équation (6.27) un délai de 2 portes non-et.

$$S_{ET} = 2 \times S_{NON-ET} \tag{6.13}$$
$$S_{OU-EX} = 4 \times S_{NON-ET} \tag{6.14}$$

$$TP_{ET} = 2 \times TP_{NON-ET} \tag{6.15}$$
$$TP_{OU-EX} = 3 \times TP_{NON-ET} \tag{6.16}$$

$$S_{H3} = b \times S_{ET} + (b-1) \times S_{OU-EX} \tag{6.17}$$

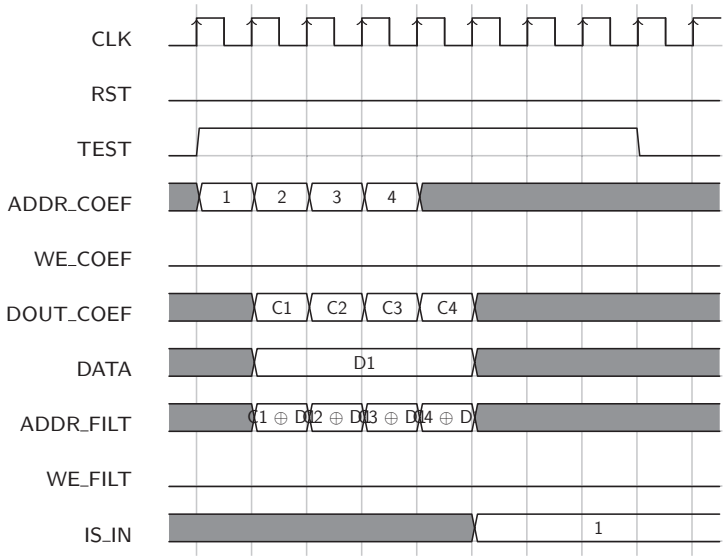

FIGURE 6.9 – Chronogrammes d'une scrutation du filtre

$$
\begin{aligned}
&= b \times (2 \times S_{NON-ET}) + (b-1) \times (4 \times S_{NON-ET}) && (6.18)\\
&= (6b-1) \times S_{NON-ET} && (6.19)\\
S_{OU-EX} &= b \times S_{OU-EX} && (6.20)\\
&= b \times (4 \times S_{NON-ET}) && (6.21)\\
&= 4b \times S_{NON-ET} && (6.22)
\end{aligned}
$$

$$
\begin{aligned}
TP_{H3} &= TP_{ET} + (b-1) \times TP_{OU-EX} && (6.23)\\
&= (2 \times TP_{NON-ET}) + (b-1) \times (3 \times TP_{NON-ET}) && (6.24)\\
&= (3b-1) \times TP_{NON-ET} && (6.25)\\
TP_{Hx} &= TP_{OU-EX} && (6.26)\\
&= 2 \times TP_{NON-ET} && (6.27)
\end{aligned}
$$

Le Tableau 6.1 montre l'implémentation post placement-routage et sans contrainte, des deux fonctions de hachage h3 et hx sur cible FPGA Spartan-6 XC6SLX45T. Les résultats en surface et en délais sont donnés pour des hachages sur 32 et 128 bits.

De ces chiffres, nous remarquons tout d'abord une similitude des deux fonctions en surface, quelle que soit la taille du mot à hacher. La complexité peut alors paraître égale, en désaccord avec les équations précédentes, si l'on ne s'intéresse qu'à l'échelle macroscopique de la LUT. Cependant, la densité des portes au sein de cet élément peut être extrêmement large. La complexité n'est donc pas visible à cette échelle mais existe bel et bien structurellement. Les chiffres des délais nous montrent que la fonction de hachage h3 propose une latence bien plus importante que la fonction hx, tout à fait en adéquation avec les équations vues auparavant. Il est important de souligner que ces délais sont, non seulement dûs au temps de propagation des signaux dans la logique, mais également dans le routage (Tableau 6.1). Si une simple fonction de hachage h3 sur quelques bits limite cet inconvénient, il n'en va pas de même pour le hachage d'une entrée large. Son utilisation est alors dissuasive dans le cadre d'une implémentation dans des systèmes pauvres en ressources et/ou le délai de propagation est un critère fort.

L'alternative qu'est la fonction de hachage hx, propose des surfaces et des

	32 bits			128 bits		
	LUTs	Slices	Délai (ns)	LUTs	Slices	Délai (ns)
Hachage h3	17	7	16.284	65	26	66.696
Hachage hx	16	7	0.308	64	28	0.308

Tableau 6.1 – Ressources et délais des fonctions de hachage h3 et hx

	Img		VoD		Com		Halg	
	UP	PP	UP	PP	UP	PP	UP	PP
AC Code	20	6.25	38	6.5	17.75	17.75	-	-
tag 64 bits								
AC Code	2560	800	4864	832	2272	2272	-	-
tag 128 bits								
AC Code	1280	400	2432	416	1136	1136	-	-

Tableau 6.2 – Mémoire requise pour le stockage des tags AC

latences extrêmement réduites. Cependant, et ce contrairement à la famille de fonctions h3, les propriétés de ce type de hachage ne remplissent pas les exigences typiquement requises. L'état interne n'est pas suffisamment mixé pour produire l'effet d'avalanche, et les permutations d'un simple ou-exclusif affectent négativement l'uniformité de la distribution.

Dans notre cas précis, les données que nous souhaitons hacher, i.e. les tags, sont issus d'opérations aléatoires cryptographiques selon les spécifications de sécurité (McGrew et Viega, 2005). En nous appuyant sur les arguments de (Mitzenmacher et Vadhan, 2008) de la Section 6.2.3.3 de ce même chapitre, nous notons que l'aspect aléatoire de ces données nous permet de conclure positivement sur la validité de notre choix.

Les fonctions de hachage sont donc désormais matériellement simplifiées puisque seul un ou-exclusif est nécessaire, opération typiquement considérée comme une primitive de la logique d'un FPGA. Une nouvelle fonction de hachage peut être obtenue en effectuant un changement des coefficients d_{ij}.

Ajoutons que les opérations ou-exclusif peuvent être implémentées dans les slices DSP du FPGA, dédiées au traitement du signal. Trois avantages sont à noter dans ce cas. Premièrement, leur utilisation limite l'emploi de la logique utilisateur. Deuxièmement, les fréquences de fonctionnement sont plus importantes que cette même logique, puisqu'il s'agit de bloc durci, et troisièmement, elles sont naturellement fondées telles qu'elles limitent le routage nécessaire avec les mémoires BRAMs.

6.3.2 Génération des coefficients de hachage

Que ce soit pour la fonction de hachage h3 ou hx la première étape essentielle avant toute utilisation du filtre est la génération des coefficients de hachage d_{ij}. Cette opération peut s'effectuer de plusieurs façons. Toujours dans l'optique de réduire au maximum la surface et la latence, notre approche passe par l'utilisation d'un composant LFSR qui est typiquement recommandé en lieu et place d'un compteur monotone ou d'un générateur de nombres aléatoires RNG. La génération peut s'appliquer à tout instant donné, soit au démarrage système, soit en cours de l'exécution.

La génération des coefficients de hachage est montrée par les chrono-

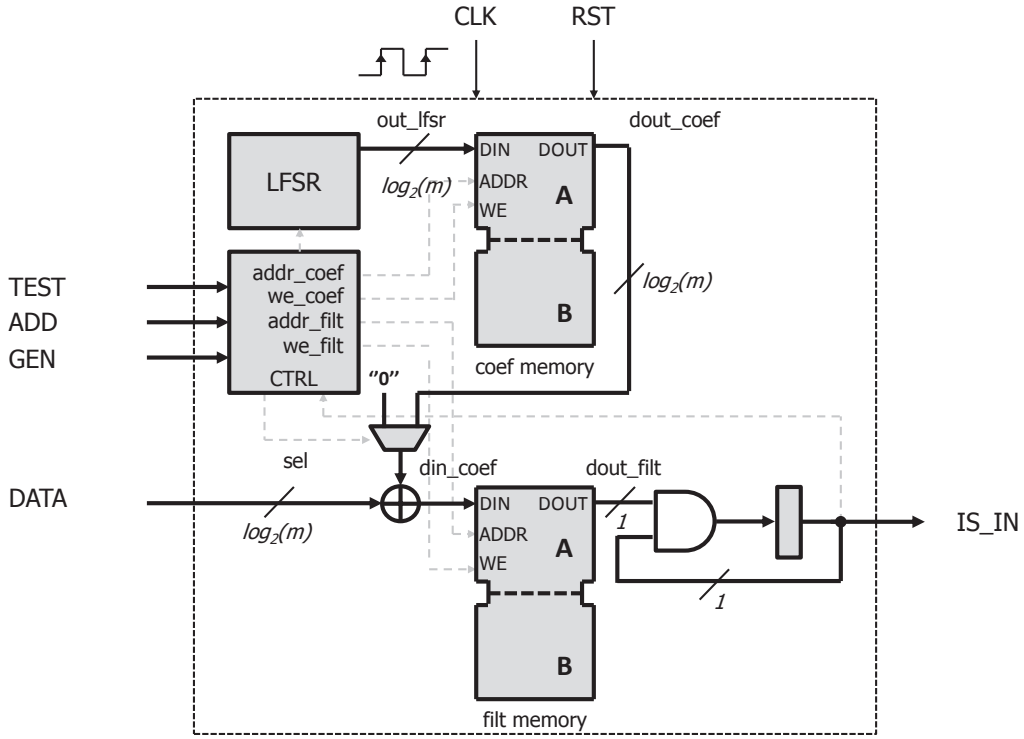

FIGURE 6.10 – Architecture matérielle d'un filtre de Bloom

grammes de la Figure 6.7, ceci pour 4 coefficients. Chacune de ces valeurs est écrite dans une mémoire associée et implémentée dans une BRAM. Rappelons que pour ce type d'élément, la lecture se fait en 1 cycle d'horloge et l'écriture, en 2 cycles. Dès lors, la génération de k fonctions de hachage peut être assimilée à la génération de k coefficients de hachage et la latence de génération est régie par l'équation :

$$T_{gen} = 2 \times k \tag{6.28}$$

Et le nombre total de bits pour le stockage de ces coefficients, se calcule par les équations :

$$\begin{aligned} Mem_{filt} &= m & (6.29)\\ Mem_{coef} &= k \times log_2(Mem_{filt}) & (6.30) \end{aligned}$$

Où m est le nombre total de bits du filtre, et se calcule par le produit du nombre d'éléments n à ajouter au filtre, et du nombre de bits pour représenter un élément. $\log_2(\text{Mem}_{filt})$ est le nombre de bits utilisés pour adresser l'espace mémoire du filtre.

6.3.3 Programmation du filtre

Deuxième étape de l'utilisation du filtre, la programmation. Avec l'aide des coefficients précédemment générés, la programmation consiste à ajouter au filtre, les éléments de l'ensemble qui seront soumis à la vérification d'appartenance, i.e. la scrutation. La Figure 6.8 décrit les chronogrammes dans le cas d'un ajout d'un élément $D1$ au filtre. La latence de la programmation d'un élément est donnée par l'équation suivante :

$$T_{prog} = 1 + (2 \times k) \tag{6.31}$$

p	m/n (bits)	m (bits)	$log_2(m)$ (bits)	k	k×$log_2(m)$ (bits)	mem total (bits)	gain (%)	BRAMs (18 Ko)	gen + prog (cycles)
tag 64 bits									
0.1	5	3988	12	4	48	4036	92.4	1	17
0.01	10	7975	13	7	91	8066	84.9	1	29
0.001	15	11963	14	10	140	12103	77.3	1	41
1E-04	20	15950	14	14	196	16146	69.7	1	57
1E-05	24	19937	15	17	255	20192	62.1	2	69
1E-06	29	23925	15	20	300	24225	54.5	2	81
1E-07	34	27912	15	24	360	28272	46.9	2	97
1E-08	39	31900	15	27	405	32305	39.3	2	109
1E-09	44	35887	15	30	480	36367	31.7	3	121
1E-10	48	39874	16	34	544	40418	24.1	3	137
1E-11	53	43862	16	37	592	44454	16.5	3	149
1E-12	58	47849	16	40	640	48489	8.9	3	161
1E-13	63	51836	16	44	704	52540	1.3	4	177
1E-14	68	55824	16	47	752	56576	-6.3	4	189
1E-15	72	59811	16	50	800	60611	-13.8	4	201
1E-16	77	63799	16	54	864	64663	-21.4	4	217
1E-17	82	67786	16	57	969	68755	-29.1	5	229
tag 128 bits									
0.1	5	1994	11	4	44	2038	96.2	1	17
0.01	10	3988	12	7	84	4072	92.4	1	29
0.001	15	5982	13	10	130	6112	88.5	1	41
1E-04	20	7975	13	14	182	8157	84.7	1	57
1E-05	24	9969	14	17	238	10207	80.8	1	69
1E-06	29	11963	14	20	280	12243	77.0	1	81
1E-07	34	13956	14	24	336	14292	73.2	1	97
1E-08	39	15950	14	27	378	16328	69.3	1	109
1E-09	44	17944	15	30	450	18394	65.5	2	121
1E-10	48	19937	15	34	510	20447	61.6	2	137
1E-11	53	21931	15	37	555	22486	57.8	2	149
1E-12	58	23925	15	40	600	24525	53.9	2	161
1E-13	63	25918	15	44	660	26578	50.1	2	177
1E-14	68	27912	15	47	705	28617	46.3	2	189
1E-15	72	29906	15	50	750	30656	42.4	2	201
1E-16	77	31900	15	54	810	32710	38.6	2	217
1E-17	82	33893	16	57	912	34805	34.6	3	229
...									
1E-26	125	51836	16	87	1392	53228	2.7	4	349

Tableau 6.3 – Paramètres, gains et pénalités, en mémoire et latence, de l'approche filtre de Bloom

6.3.4 Scrutation du filtre

La scrutation est l'étape permettant de vérifier l'appartenance d'un élément au filtre sous couvert d'un taux de faux positifs. Elle prend place lorsque les coefficients de hachage sont générés et que le filtre est programmé. La Figure 6.9 expose les chronogrammes de l'étape de scrutation d'un élément $D1$, pour vérifier sa présence dans le filtre La latence de la scrutation d'un élément est donnée par l'équation suivante :

$$T_{scrut} = 1 + k \tag{6.32}$$

L'architecture complète de notre filtre de Bloom optimisée pour les systèmes embarqués, est montrée en Figure 6.10. Elle est composée d'un LFSR pour la génération des coefficients, de deux mémoires, l'une pour les coefficients de hachage (coef memory) et l'autre pour les éléments du filtre (filt memory), d'un ou-exclusif, utilisé pour le hachage hx, et d'un bloc logique de contrôle (CTRL). En sortie du filtre, une simple porte et à deux entrées et un registre permettent de connaître l'état actuel de scrutation d'un élément.

6.4 Approche expérimentale et résultats

Pour visualiser plus clairement l'impact favorable que peut avoir un filtre de Bloom sur le stockage des tags cryptographiques, nous considérons les 4 applications multi-tâches du Chapitre 4 à savoir, l'application image (Img), vidéo à la demande (VoD), communication (Com) et algorithmes de hachage (Halg). Parmi ces 4 applications, nous souhaitons étudier l'espace de compression que nous procure les filtres de Bloom, dans le cas particulier de l'application VoD, avec la politique de sécurité programmable. La construction du filtre de Bloom ne supportant pas la suppression d'élément, nous nous intéressons aux tags du code de l'application uniquement. Le Tableau 6.2 rappelle la mémoire, en octets, qu'il faut allouer pour les tags issus du code de l'application. Il marque aussi l'équivalence en nombre de tags de 64 bits et de 128 bits.

Pour notre expérimentation, le nombre de tags à stocker est de 832 lorsque les tags sont de tailles 64 bits, et de 416 quand ils sont de tailles 128 bits. Le Tableau 6.3 dresse les paramètres, les gains et les pénalités, en mémoire et en latence obtenus avec une représentation par filtre de Bloom, pour ces deux tailles de tags. p est la proportion de faux positifs. m est la taille du filtre en nombre de bits. La valeur m/n dicte le nombre de bits pour représenter un élément du filtre. $log_2(m)$ est le nombre de bits utilisés pour adresser l'espace mémoire du filtre. C'est aussi la taille des coefficients, donc la dimension du composant LFSR et du ou-exclusif pour le hachage. La valeur de k indique le nombre d'opération de hachage nécessaire et agit directement sur la latence du circuit lors de la génération, la programmation et la scrutation. Cette valeur influe sur la taille totale du filtre d'un nombre de bits de $k \times log_2(m)$. $memtotal$ est la taille totale du filtre en bits. Le $gain$ en mémoire est relevé en %. $BRAMs$ est le nombre de blocs mémoire de 18 Ko nécessaire pour le stockage des coefficients et des éléments du filtre. $gen+prog$ est la sommation de la latence, en cycles, pour la génération des coefficients de hachage et la programmation du filtre, pouvant être effectuées hors-ligne.

Sans filtre de Bloom, 64×832 ou 128×416 bits mémoire doivent être disponibles pour la sauvegarde des tags, soit 53248 bits. C'est à dire que 4 BRAMs de 18 Ko sont nécessaires. A ces 4 BRAMs, il faut ajouter la logique permettant la recherche dans ces mémoires (tables associatives), puisque les tags, dont il est question ici, peuvent potentiellement se trouver n'importe où dans cet espace alloué.

Le filtre de Bloom permet d'explorer d'autres possibilités. Dépendamment des conditions du concepteur, un tag peut être représenté par un nombre arbitraire de bits. D'après le Tableau 6.3, si 5 bits sont choisis pour la représentation, 3988 bits pour le filtre sont nécessaires, 48 bits pour le stockage des coefficients de hachage, pour un total de bits mémoire de 4036, soit 1 BRAM de 18 Ko. Ceci est équivalent à un gain réel de 92.4% sur une proposition classique, sans filtre. La latence de génération des coefficients et de la programmation du filtre est de 8 et 9 cycles d'horloge respectivement, soit 17 au final. La scrutation d'un élément s'effectue, elle, en 5 cycles. La proportion de faux positifs notée, est de 10%. Le Tableau montre des gains négatifs dés que l'on dépasse la représentation initiale de 64 bits. Pour des tags de 128 bits, nous relevons des gains en mémoire de 96.2% pour une représentation sur 5 bits, les autres paramètres étant strictement similaires pour cette valeur.

Rajoutons qu'à l'inverse de l'approche classique, les filtres de Bloom permettent une recherche induite dans la structure, ce qui limite d'autant plus, la logique

Il est aisé de voir que la solution réduit la mémoire impérative de façon avantageuse. Toutefois, et c'est là le compromis dont il faut tenir compte, la taille de la représentation implique un taux de faux positifs. On a donc "une" chance plus ou moins élevée, de considérer un tag comme existant, alors qu'il ne l'est pas. Ceci peut être problématique pour la sécurité du système. La première colonne du Tableau 6.3 donne intuitivement le niveau de sécurité contre une attaque par force brute. Ainsi, des niveaux de sécurité s'échelonnants de $\frac{1}{10}$ jusqu'à $\frac{1}{1E-11}$ pour des tags de 64 bits et de $\frac{1}{10}$ jusqu'à $\frac{1}{1E-24}$ pour des tags de 128 bits. Ces ordres de grandeur sont faibles comparativement aux approches des Chapitres 3 et 4 ($\frac{1}{2^{32}}$, $\frac{1}{2^{64}}$, $\frac{1}{2^{32}}$). Le filtre de Bloom est typiquement une représentation du compromis à faire entre latence, mémoire et sécurité. Le concepteur peut agir sur les paramètres p, m, n et k pour obtenir un niveau de sécurité approprié mais il est cependant très clair que le filtre ne s'applique pas lorsqu'un niveau élevé est requis. Dans ce cas, d'autres méthodes de compression sans perte devraient être considérées.

6.5 Conclusions

La structure de filtre de Bloom proposée dans ce chapitre, est une approche neuve pour les systèmes embarqués sécurisés. Tout comme les propositions précédentes, les implémentations dessinent de faibles pénalités en latence et en mémoire. Pour la première fois, le matériel cryptographique très dense que représente les tags est représenté efficacement, pesant énormément dans le nombre total d'éléments mémoires devant être ajoutés. Par rapport aux approches classiques, des taux très importants de compression sont visibles. Un coût réduit, pour une proposition économiquement viable est la clé de ces travaux, où les faux positifs se doivent cependant d'être attentivement étudiés, au cas par cas.

Conclusions et Perspectives

Tout au long de ce document, nous avons émis des propositions pour la sécurité haut-débit. Les différentes contributions matérielles montrent qu'il est possible de fournir des solutions efficaces pour les systèmes embarqués qui ont pourtant une restriction en ressources importantes. En résumé, voici nos contributions :

1. ***Chapitre 2*** *: une revue et étude complète des différentes primitives de sécurité adaptées au haut-débit et spécifique aux systèmes embarqués.*

2. ***Chapitre 3*** *: une implémentation clé-dépendante robuste de la fonction d'authentification GHASH, avec mise à disposition des codes sources.*

3. ***Chapitre 4*** *: une proposition avec implémentations à l'appui, d'améliorations pour la sécurité des systèmes embarqués qui se base sur la sécurité programmable.*

4. ***Chapitre 5*** *: une plate-forme de reconfiguration dynamique partielle, implémentée de bout-en-bout, et pouvant servir de support pour la sécurité configurable.*

5. ***Chapitre 6*** *: l'application de la structure filtre de Bloom pour stocker efficacement le matériel cryptographique spécifique que sont les tags. L'implémentation est adaptée aux systèmes embarqués.*

6. ***Annexe*** *: la développement (en cours) de l'approche du Chapitre 4 mais pour une architecture avec de multiples coeurs de processeur.*

A partir de là, nous voyons distinctement quelques directions qui pourraient être intéressantes de suivre.

Comme souligné dans le Chapitre 3, notre approche clé-dépendante de la fonction GHASH est bénéficiaire si un changement de clé de hachage survient infréquemment. Dans le cas où le changement de clé est une condition bloquante, l'usage de la construction HMAC (Association, 2000) (Krawczyk *et al.*, 1997b) est particulièrement intéressante. HMAC peut en effet être utilisée en combinaison avec une fonction de hachage cryptographique comme GHASH et est une encapsulation sécurisée de clés. La structure clé-dépendante de GHASH peut alors être relâchée partiellement, sans augmenter la surface et sans diminuer la fréquence de fonctionnement, pour peu que deux circuits possèdent une clé unique. Non content de proposer des propriétés cryptographiques, la qualité de la fonction GHASH peut être envisagée pour réaliser du hachage simple, rapide et robuste, en un mot, haute performance. Une évaluation d'implémentations logicielles pourrait aussi proposer des résultats intéressants.

Dans le Chapitre 4, les politiques de sécurité configurables sont simples mais ne présentent pas d'états intermédiaires. Peu de publications se proposent de construire formellement ces règles. Pour notre part, la logique floue semble constituer une opportunité pouvant permettre de les faire apparaître et pourrait être utilisée pour fournir au concepteur une gamme flexible de politiques. Comme vu dans l'extension qu'est l'Annexe, l'approche multicoeur pourrait largement bénéficier de ces politiques étendues. Plusieurs autres opportunités subsistent pour de futurs travaux. Notre approche pourrait être évaluée pour supporter de l'authentification seulement, en plus des trois niveaux décrits ici. Nos travaux peuvent être également étendus pour cibler des technologies ASIC en considérant une distribution de clés sécurisée et une taille de tag et de stockage SMM figées. Une analyse détaillée de la taille de ces deux éléments pour une variété d'application devrait alors être effectuée. Une optimisation supplémentaire serait de stocker une partie de tous les TS et les valeurs de tag hors-puce. Un mécanisme plus complexe serait alors nécessaire pour assurer que l'information n'est pas compromise par une attaque.

Autour du Chapitre 5, il est envisageable de bâtir d'autres optimisations pour les systèmes reconfigurables. En ce qui concerne l'aspect réseau tout

d'abord. Les adresses IP sont, en effet, assignées de façon statique. Il serait bon de pouvoir le faire automatiquement et dynamiquement en implémentant un serveur de configuration automatique des paramètres IP (Dynamic Host Configuration Protocol, DHCP). L'ajout d'un nouveau client serait alors simple, et sans intervention manuelle. La qualité de service QoS est tout aussi essentielle pour LAN et WLAN. Nombre de travaux (Grewal et DeDourek, 2004), (RAKNET) inclut une couche supplémentaire (logicielle et/ou matérielle) au dessus de UDP pour assurer la transmission de données en implémentant des algorithmes simples. Ce serait alors une bonne solution de se concentrer sur ces méthodes lorsque des pertes de paquets sont possibles, par exemple dans un environnement hostile. Par la suite, un portage sur des processeurs soft-core Microblaze serait également une idée intéressante. Comme le Virtex-2 est un FPGA considéré cible dépréciée, nous pouvons envisager une migration vers des FPGAs de famille supérieure, Virtex-5 ou Virtex-6. Enfin, il est également à noter que la sécurité des transactions du protocole n'est pas abordée ici et est également un chemin à suivre dans la volonté de posséder une chaîne de confiance complète.

Concernant le Chapitre 6 plusieurs améliorations peuvent être envisagées. Comme mentionné précédemment, les filtres de Bloom standards ne peuvent supporter la suppression d'élément. Cette insuffisance est paliée par l'ajout d'un compteur pour chaque élément de la structure (Whang *et al.*, 1990) (Fan *et al.*, 2000). Le fonctionnement est similaire au filtre de Bloom classique excepté qu'il est possible de traquer les insertions et les suppressions. Lorsqu'un élément est inséré, le compteur lui correspondant est incrémenté. Inversement, il est décrémenté lorsqu'un élément est supprimé du filtre. Une autre option pour la suppression est le filtre de Bloom supprimable (Rothenberg *et al.*, 2010). Le coût en mémoire pour supporter cette caractéristique est ici minimal et n'introduit pas de faux négatifs. La structure est basée sur l'idée de garder un enregistrement des régions de bits où des collisions ont eut lieu et d'exploiter la notion que les éléments peuvent être effectivement supprimés si au moins l'un des bits est à zéro. La suppression d'élément est alors probabiliste et dépend de la taille du filtre. Compresser un filtre de Bloom (Mitzenmacher, 2001) améliore également les performances lorsqu'un tel filtre est passé comme message entre des noeuds distribués. Lorsque de l'information doit être transmise répétitivement et que la bande passante est un facteur limitant, cette option est largement envisagée. Il est à noter que la compression peut également participer à la réduction du taux de faux positifs

comparée à un filtre de Bloom non compressé.

D'autres perspectives semblent également émerger. La virtualisation des systèmes d'exploitations est un sujet en vogue actuellement, et spécialement pour les systèmes embarqués. Au lieu d'avoir un processeur associé avec un OS, plusieurs OS s'exécutent sur ce même processeur ce qui peut offrir des perspectives pour la sécurité. En effet, en ayant deux systèmes d'exploitation, le processeur peut exécuter des applications à partir de deux espaces mémoires différents. Grâce à cela, un système d'exploitation non sécurisé ne peut pas accéder aux données de celui qui l'est. Quoiqu'il en soit, le besoin de la protection de la mémoire est toujours d'actualité et les communications entre les deux OS doivent être étudiées.

Ce sont des challenges qui font apparaître des perspectives pour la sécurité mais aussi pour les attaquants. La sécurité sur FPGA étant un champ d'expérimentation relativement jeune, gageons que ceux-ci deviendront de plus en plus capables de supporter d'importants dispositifs de sécurité, mais notre volonté s'inscrit fortement dans le but de fournir des solutions de sécurité pratiques pour les systèmes embarqués qui s'appuient sur des circuits reconfigurables.

Annexe A : Une Approche Multicoeur

A-1 Introduction

Un système sur-puce n'est désormais plus limité à un seul coeur de processeur. Multicoeurs et pluricoeurs deviennent la norme par les récents progrès de l'informatique à très grande échelle. Celle-ci se veut le fer de lance de l'optimisation de la gestion de tâches et promet moult progrès : applications web plus rapides, services de cloud computing plus réactifs, infrastructures sans-fil à moindre coût et même une sécurité renforcée. Pourtant, les propositions autour de cette dernière propriété se comptent, pour le moment, sur les doigts de la main. Largement d'aspect expérimental, ce chapitre est l'extension du Chapitre 4 sur la sécurité configurable dans les systèmes embarqués et explore quelques résultats préliminaires d'implémentation sur une architecture multicoeurs.

Cette annexe est partionnée en quatre parties et est organisée comme suit. La première partie revient sur les propositions multicoeurs issues de travaux académiques. La seconde est axée sur notre banc de test logiciel adapté pour évaluer nos architectures proposées dans la troisième partie. Enfin les résultats d'implémentation sur FPGA sont donnés en quatrième partie et sont suivis de la conclusion.

Les résultats actuels ne nous permettant pas de statuer clairement sur l'ensemble de nos propositions, la totalité des mesures relevées se doit d'être prise avec des pincettes. C'est dans un souci de montrer clairement toutes nos contributions que cette annexe est présente, elle doit donc être considérée comme telle.

A-2 Propositions académiques d'architectures multicoeurs sécurisées

Les chercheurs ont exploré les conceptions multicoeurs sécurisées. Cependant la complexité ajoutée à ce type de système rend leurs évaluations difficiles. Dans cette section nous décrivons quelques propositions qui sont données à titre d'information.

Le schéma de sécurité pour architecture multicoeurs proposé par (Shi *et al.*, 2004) se base sur la technique de l'écoute de bus (bus snooping)

pour la cohérence de cache. L'architecture de base est de type signature-et-vérification comme beaucoup des systèmes monoprocesseurs. Ils proposent deux domaines de sécurité avec une frontière située sur le contrôleur mémoire NorthBridge. Toutes les données entrantes, incluant le code exécutable, sont chiffrées et signées avec des clés vendeurs (vendors keys). Le contrôleur mémoire déchiffre et vérifie les données et les chiffre et les signe de nouveau avec des clés systèmes (system keys). Au travers du système,tout le matériel cryptographique et notamment les clés contenues dans les différentes puces doivent correspondre. Les données sont déchiffrées et vérifiées lorsqu'elles sont échangées avec une puce en particulier. Un tampon d'authentification séquentiel est utilisé pour autoriser l'exécution spéculative en parallèle de la vérification des données. Les auteurs revendiquent une pénalité en performance de 5% sur la suite de benchmarks SPLASH2.

Dans le cas des travaux de Zhang, (Zhang *et al.*, 2005) les auteurs ciblent également un système multiprocesseurs avec écoute de bus. Ils supposent qu'une méthode existe pour sécuriser la mémoire externe (telle que signature-et-vérification) et se concentrent sur les messages utilisés pour la cohérence de cache. Les processeurs sont divisés en groupes avec un unique identifiant ID. Les messages de chaque processus sont étiquetés avec les identifiants de groupe de processus ce qui nécessite des lignes supplémentaires sur le bus. Tous les messages sont chiffrés avec des masques jetables et signés avec le mode CBC-MAC. Un compteur de message global est utilisé pour sérialiser les messages, et chaque processeur possède un buffer circulaire qui pré-calcule les masques jetables pour un chiffrement et déchiffrement rapide. Pour un message donné, un processeur fournit la signature et les autres recalculent la signature et vérifient le message individuellement. Pour un maximum de sécurité, chaque message est vérifié. La nature chaînée du schéma CBC, peut être utilisée pour vérifier des lots de messages ce qui améliore les performances. Leurs simulations prédisent que les messages protégés de cette manière ajoutent une pénalité de 2.03% en plus de la pénalité requise pour protéger la mémoire externe. Le trafic du bus augmente également de 34%.

L'approche de Lee, (Lee *et al.*, 2007), adresse la protection des messages de cohérence de cache dans un système distribué à mémoire partagée. Leur but est de fournir de la sécurité quel que soit le système d'interconnexion. Ils utilisent le mode GCM avec une seule autorité qui assigne des plages de compteur aux processeurs. Lorsqu'un processeur reçoit un compteur qui lui

Application	Taille du code (octets)	Taille des données (octets)	Paramètre étalon	Valeur du paramètre
CRC	21550	8678	Taille du message	4096
Matrix	16058	15850	Taille de la matrice	32
LU	23142	21990	Taille de la matrice	48
Choleski	25352	36324	Taille de la matrice	64
FFT	26644	10396	Nombre de points	512
DFT	35560	11748	Nombre de points	512
FIR	22222	3862	Nombre de taps, Taille du noyau	15, 8
Radix	16178	4586	Taille du tableau à trier	256
WHT	15666	3578	Taille du vecteur	8
N-body	27592	6916	Nombre de particules	42

Tableau A-1 – Empreinte mémoire et paramètres associés aux applications de test

est affecté, il pré-calculent les masques nécessaires pour GCM et les stocke dans une file pour accélérer le chiffrement des messages sortants. Les autres processeurs pré-calculent ces mêmes masques et les cachent pour accélérer le déchiffrement des messages entrants. Les masques récemment utilisés sont également cachés au cas où un bloc est reçu et est envoyé de nouveau sans modification. Les auteurs annoncent une pénalité moyenne de 4% pour protéger les messages de cohérences et admettent une faiblesse liée aux messages de contrôle qui ne sont pas protégés.

Pour Patel, (Patel *et al.*, 2007), l'utilisation d'un moniteur pour assurer l'intégrité des programmes qui s'exécutent sur un système-sur-puce-multiprocesseurs (Multiprocessor System-on-Chip, MPSoC) est une solution. Le compilateur cartographie les différents chemins d'exécution de la partie critique du code et génère une base de donnée de contraintes des chemins valides et des temps minimums et maximums autorisés pour l'exécution de chaque bloc. A l'exécution, un processeur est utilisé comme moniteur pendant que les autres exécutent les programmes. Chaque processeur rapporte son flot et son temps d'exécution au moniteur avec des FIFOs. Le processeur-moniteur vérifie ces données avec la base de contraintes. Si le flot n'est pas valide, ou si le bloc prend trop ou trop peu de temps pour s'exécuter, alors le programme a été compromis. Cette approche a plusieurs faiblesses admises notamment la dépendance d'une analyse statique du code programme qui peut s'avérer non précise pour profiler les chemins d'exécution avec dépendances de données. Il n'adresse pas la confidentialité du code et ne protège pas les données. Les pénalités en performances s'étendent de 6.6% à 9.3%.

Enfin, l'équipe du GA Tech-NCSU dans (Rogers *et al.*, 2008) étend leur conception précédente dans le domaine multiprocesseurs qui adresse les systèmes distribués à mémoire partagée. Dans leur proposition, chaque processeur maintient un arbre pour une protection dynamique des données. Des nombres de séquence et horodatage sont communiqués entre processeurs avec les blocs de données et leurs signatures. Les messages de cohérence qui contiennent des blocs de données sont aussi protégés par un message de signature additionnel. La vérification n'est cependant pas précise ce qui peut mener à des vulnérabilités de sécurité.

Dans cette annexe, nous proposons un aperçu de nos premiers résultats d'implémentations des travaux issus du Chapitre 4, pour des conceptions à

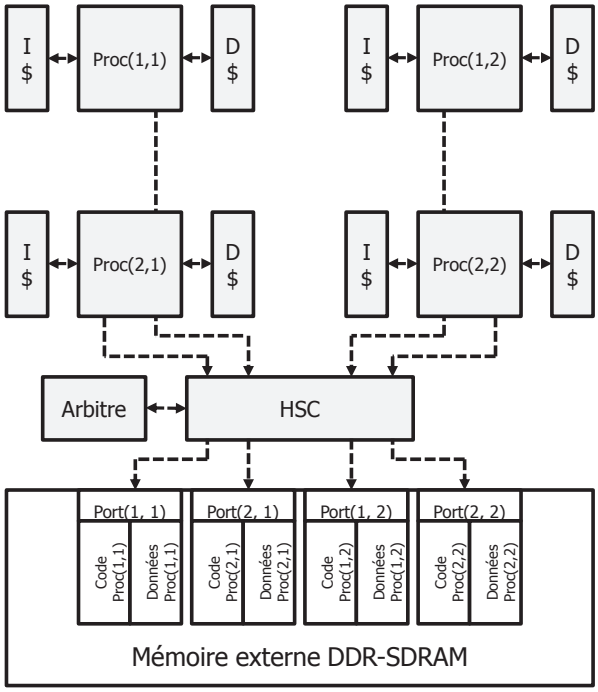

FIGURE A-1 – Grille typique de 4 coeurs de processeur avec un coeur matériel de sécurité HSC

base de multiple coeurs de processeur. Il est nullement question, et ce serait d'ailleurs précipité que de le faire, de se comparer aux travaux précédents.

A-3 Banc de test utilisé pour l'évaluation

Pour valider notre première approche, 9 applications étalons ont été mises au point.

- **CRC** : Le CRC permet de détecter des erreurs de transmissions en ajoutant des informations redondantes. Des sommes de contrôle, sont calculés avant un transmission et après une transmission sur un canal généralement considéré comme bruité. La comparaison des deux valeurs calculées permet de s'assurer que la transmission est correcte.

- **Multiplication de matrice** : De la théorie des jeux aux théories issues de l'économie, de la fouille de texte à la compilation automatisée de thésaurus, le produit matriciel est largement employé dans bien des domaines. Un exemple très commun d'application est celui des mathématiques pour la résolution d'équations linéaires.

- **Décomposition LU** : La décomposition LU est une méthode issue de l'algèbre linéaire qui décompose une matrice en un produit de deux matrices triangulaires, l'une inférieure (L) et l'autre supérieure (U). Elle est employée pour la résolution des systèmes d'équations linéaires, pour inverser des matrices ou encore pour calculer un déterminant.

- **Factorisation de Choleski** : Utilisée pour l'évaluation d'intégrale par simulation de Monte Carlo ou pour résoudre des systèmes d'équations linéaires. André-Louis Choleski proposa une factorisation efficace capable d'obtenir des performances doublées comparativement à la décomposition LU.

- **DFT** : La transformée de Fourier discrète (Discrete Fourier Transform, DFT) est issue de la démonstration de Joseph Fourier sur la décomposition de tout signal continu en somme de sinusoïdes. Seule la transformée dite discrète est utilisée en pratique car les signaux sont échantillonnés et quantifiés.

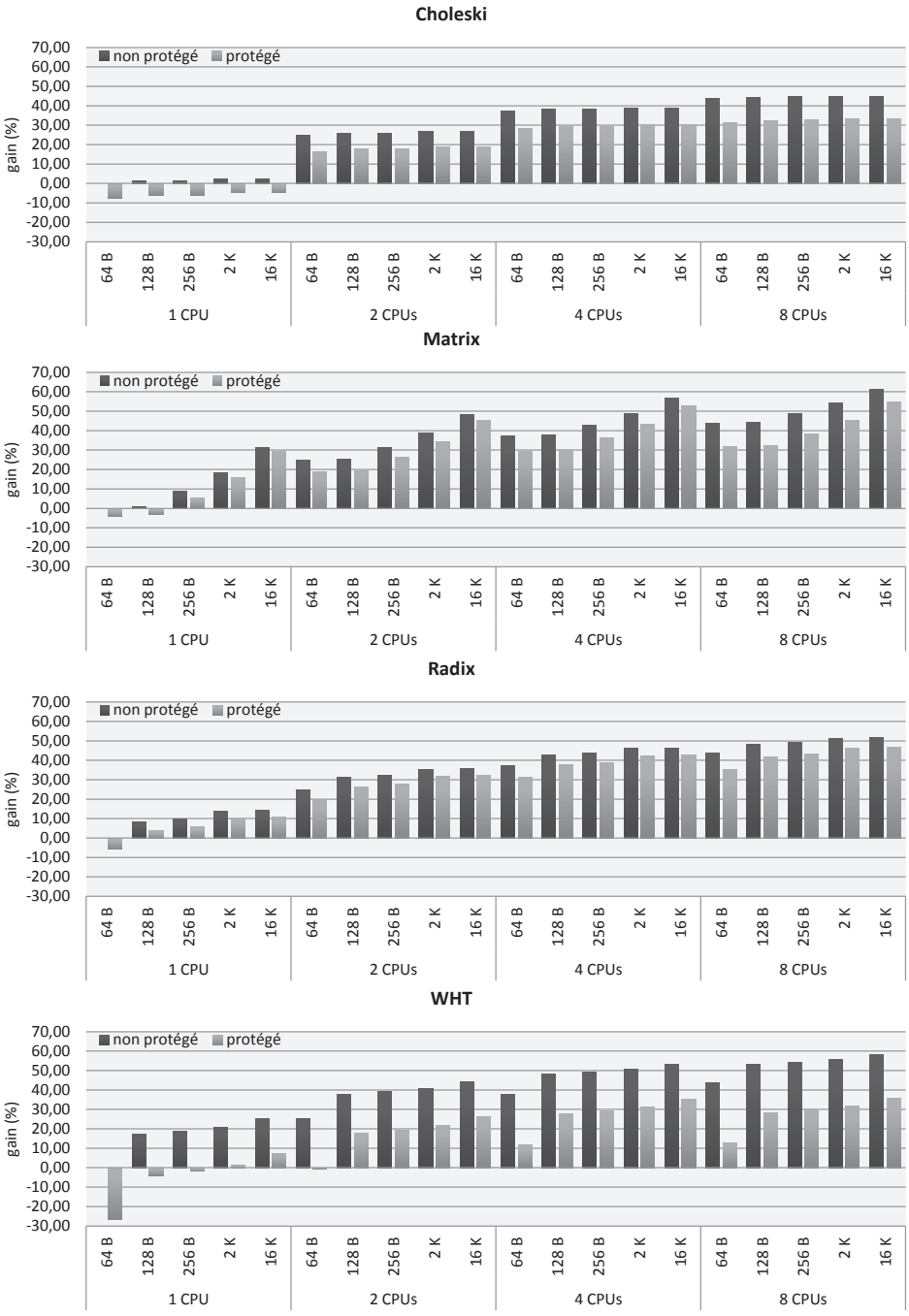

FIGURE A-2 – Gains en pourcentage des applications sans et avec protection, pour la politique write-through

- **FFT** : La transformée de Fourier rapide (Fast Fourier Transform, FFT) dite de Cooley-Tukey ou radix-2, est l'amélioration de l'algorithme en $O(n^2)$ découvert par Gauss. Elle se base sur la factorisation récursive de la formule.

- **Filtre FIR** : Les filtres à réponse impulsionnelle finie (Finite Impulse Response, FIR) sont typiquement utilisés pour supprimer des fréquences indésirables. Ce sont des éléments essentiels en communication numérique qui sont aussi extrêmement employés en radio logicielle. Dans ce domaine, les filtres sont ajustables avec de bonnes réjections et sans changement matériel.

- **Tri Radix** : Autrefois utilisé pour trier des cartes perforées, le tri radix est un tri rapide avec un temps d'exécution de $O(nk)$ ou n est le nombre d'éléments et k leur taille. Cet algorithme est aussi appelé tri par base.

- **WHT** : la transformée de Walsh-Hadamard est une généralisation de la transformée de Fourier. Elle décompose un signal en un ensemble de formes d'ondes orthogonales et rectangulaires appelées fonctions de Walsh. Elle est utilisées dans l'analyse de spectre en puissance, pour le filtrage, la résolution d'équations non-linéaires différentielles ou la conception logique et l'analyse.

- **Simulation N-body** : Le problème classique N-body consiste à simuler l'évolution de N-corps lorsque la force exercée sur chacun d'eux se pose par l'interaction avec d'autres corps du système. Les algorithmes N-body ont de larges applications comme l'astrophysique ou la dynamique moléculaire par exemple. Une simulation type s'effectue par pas de temps, lors desquelles les forces de chaque corps sont calculées. Leurs positions et divers attributs sont alors mis à jour. Si toutes les forces sont calculées directement, une nombre d'opération en complexité $O(N^2)$ est nécessaire à chaque pas de simulation.

Le Tableau A-1 décompose pour chaque application de test, la taille du code et des données en octets, le paramètre étalon et sa valeur par défaut.

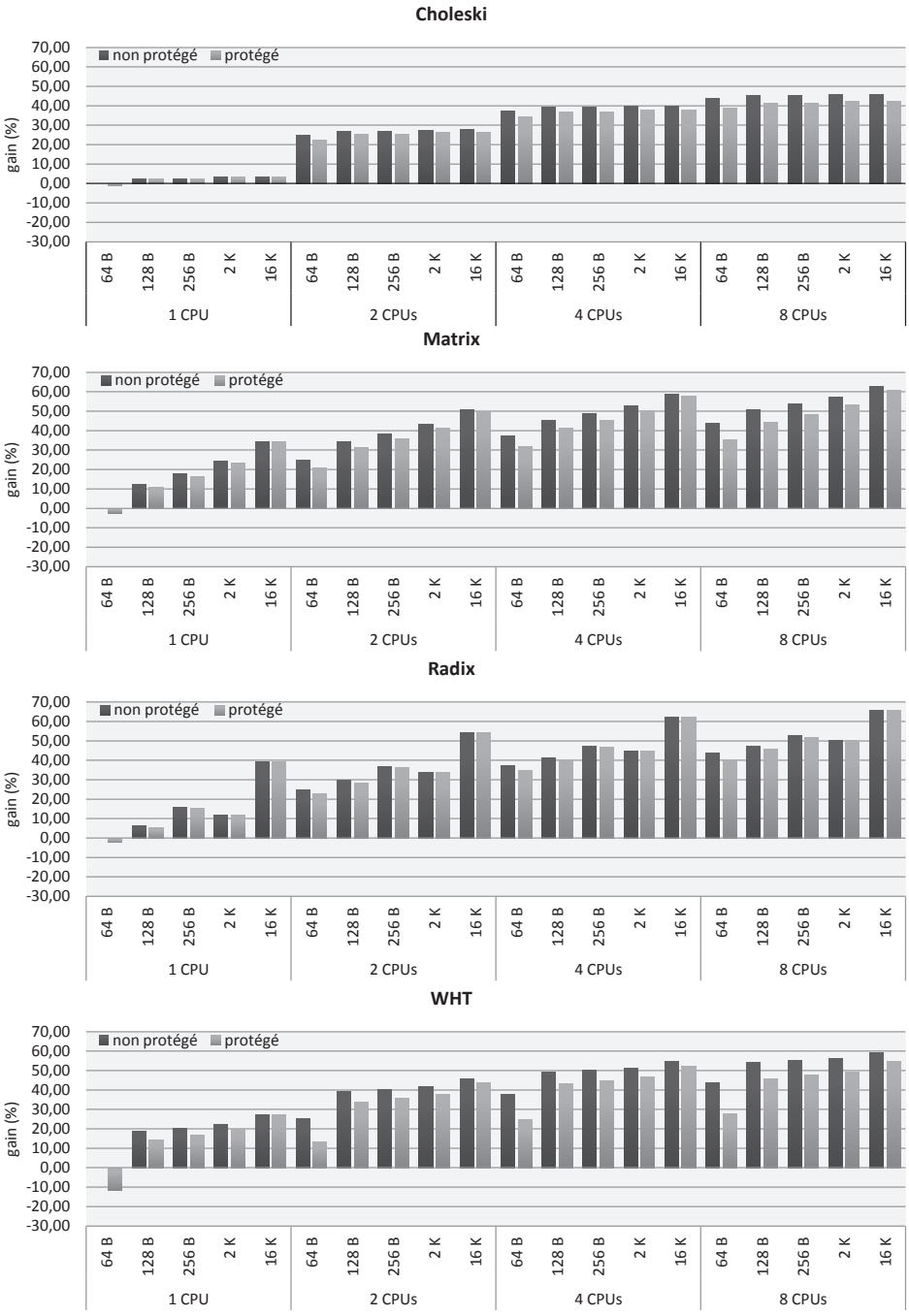

FIGURE A-3 – Gains en pourcentage des applications sans et avec protection, pour la politique write-back

A-4 Approche et résultats expérimentaux

Pour évaluer la validité de notre approche, un système basé sur FPGA ayant une structure similaire à celle décrite dans le Chapitre 4 à été prototypée. La Figure A-1 montre la grille typique de 4 coeurs connectés à une mémoire externe DDR-SDRAM contenant le code et les données de chaque coeur, via un coeur de sécurité HSC (Chapitre 4).

Notons dans un premier temps, que seuls les résultats d'expérimentation des applications Choleski, Matrix, Radix et WHT sont reportés dans cette annexe. Les autres ayant fourni des résultats très proches (par la proportion de lecture et d'écriture en mémoire externe), voir similaires.

Pour ces expériences, seule la politique de sécurité uniforme a été choisie. Gageons que l'implémentation des politiques de sécurité configurables du chapitre précédent auront un impact favorable sur la réduction de la latence des calculs cryptographiques et du nombre de tags à stocker. Avec les filtres de Bloom, il sera, normalement, possible d'accroître d'autant plus les performances de stockage.

L'expérimentation évalue des architectures avec 1, 2, 4, et 8 coeurs de processeurs. Chacun de ces coeurs possède un cache instruction (I$) fixé à 2 Ko, et cache de données (D$) de taille allant de 64 octets jusqu'à 16 Ko avec des lignes de 4 mots de 32 bits. On lui associe également les mémoires TS et AC comme vu au chapitre 4. Le contrôleur RAM Xilinx ayant cette capacité, chacun des coeurs est connecté à la mémoire externe DDR-SDRAM par un port qui lui est dédié. Le système est contraint par une fréquence utilisateur de 66 MHz.

Les gains en temps d'exécution en %, provoqués par l'ajout de coeurs par rapport à une architecture avec un seul coeur de processeur, sont donnés par les Figures A-2 et A-3. Elle évoque également la pénalité que provoque l'approche sécurisée sur une architecture non sécurisée. Le Tableau A-2 donne la pénalité en surface des implémentations, sur un FPGA Spartan-6 XC6SLX45T et sans coeur de sécurité, en nombre de registres, LUTs, slices et BRAMs pour les politiques de cache write-through et write-back. Le Tableau A-3 donne ces mêmes pénalités mais avec, en sus, un coeur de sécurité matériel basé sur AES-GCM qui est décrit dans le Chapitre 4. Les aspects

	Write-through				Write-back			
	FFs	LUTs	Slices	BRAMs	FFs	LUTs	Slices	BRAMs
1 cores 2K, 64B, 4	1188	1575	587	7	1238	1685	595	6
1 cores 2K, 128B, 4	1187	1570	558	7	1243	1699	573	6
1 cores 2K, 256B, 4	1186	1575	591	7	1245	1744	682	6
1 cores 2K, 2K, 4	1162	1540	538	8	1237	1689	641	8
1 cores 2K, 16K, 4	1163	1549	539	15	1238	1684	587	16
1 cores 2K, 64K, 4	1163	1551	606	42	1268	1704	645	43
2 cores 2K, 64B, 4	2226	2986	1110	14	2326	3242	1183	12
2 cores 2K, 128B, 4	2224	2994	1101	14	2336	3252	1210	12
2 cores 2K, 256B, 4	2222	2978	1035	14	2340	3318	1208	12
2 cores 2K, 2K, 4	2174	2936	1106	16	2324	3207	1133	16
2 cores 2K, 16K, 4	2176	2932	1070	30	2326	3231	1209	32
2 cores 2K, 64K, 4	2176	2934	1141	84	2386	3255	1310	86
4 cores 2K, 64B, 4	4300	5807	2031	28	4500	6305	2184	24
4 cores 2K, 128B, 4	4296	5807	1981	28	4520	6304	2160	24
4 cores 2K, 256B, 4	4292	5814	2172	28	4528	6431	2214	24
4 cores 2K, 2K, 4	4196	5693	1984	32	4496	6319	2239	32
4 cores 2K, 16K, 4	4200	5704	1972	60	4500	6297	2183	64
4 cores 2K, 64K, 4	-	-	-	-	-	-	-	-
8 cores 2K, 64B, 4	8448	11446	4177	56	8848	12286	4010	48
8 cores 2K, 128B, 4	8440	11385	4026	56	8888	12350	4174	48
8 cores 2K, 256B, 4	8432	11357	3888	56	8904	12638	4227	48
8 cores 2K, 2K, 4	8240	11161	3859	64	8840	12330	4153	64
8 cores 2K, 16K, 4	-	-	-	-	-	-	-	-
8 cores 2K, 64K, 4	-	-	-	-	-	-	-	-

Tableau A-2 – Résultats de synthèse des architectures avec politique write-through et write-back

cohérences de cache ne sont pas pris en compte.

Indépendamment de la sécurité, il est à souligner que bien que la politique de cache Write-back est généralement plus complexe à implémenter, les résultats en termes de performances, ramenés au coût en surface, montrent qu'une implémentation write-through est moins avantageuse qu'une politique de type write-back :

- **Politique Write-through** : l'écriture est faite de façon synchrone dans le cache et la mémoire de stockage.

- **Politique Write-back** : l'écriture n'est faite que dans le cache et un bloc de cache qui est modifié est écrit, avant d'être remplacé, dans la mémoire de stockage.

Notons, que bien que les courbes mentionnent les performances pour les architectures avec 4 coeurs et 64 Ko de cache de données, ainsi que 8 coeurs avec 16 Ko et 64 Ko de caches de données, elles sont issues d'extrapolations avec la loi de Amdahl. Elle ne sont donc pas implémentées, dû à la surface disponible du FPGA.

Nous prenons comme exemple l'application WHT, puisque c'est celle qui est la plus pénalisée par l'ajout de l'approche de sécurité. La Figure A-2, montre un impact sur les performances de -28% pour un système monocoeur avec un cache de données de 64 octets et une politique de cache write-through. Cette chute est amortie si des caches de tailles supérieures sont choisis. Pour les architectures basées sur 2, 4 et 8 coeurs, l'on distingue que le parallélisme influe positivement sur les performances. La pénalité de la sécurité est alors intuitivement réduite (-1% pour 2 coeurs avec cache de données de 64 octets) comparativement à une architecture avec un seul coeur de processeur. Cette amélioration est à relativiser si l'on compare les performances avec 2 coeurs, sans sécurité. La Figure A-3 évalue de façon similaire, les performances avec la politique write-back. Elle montre que dans les même conditions de test, l'impact sur les performances est bien moins important. L'on relève une agilité en baisse de 11% pour un seul coeur de processeur avec cache de données de 64 octets.

Comme l'on s'en doute, l'augmentation de la taille du cache de données,

	Write-through				Write-back			
	FFs	LUTs	Slices	BRAMs	FFs	LUTs	Slices	BRAMs
1 cores 2K, 64B, 4	3814	4867	1657	23	3864	4977	1665	22
1 cores 2K, 128B, 4	3813	4862	1628	23	3869	4991	1643	22
1 cores 2K, 256B, 4	3812	4867	1661	23	3871	5036	1752	22
1 cores 2K, 2K, 4	3789	4832	1608	24	3863	4981	1711	24
1 cores 2K, 16K, 4	3789	4841	1609	31	3864	4976	1657	32
1 cores 2K, 64K, 4	3789	4843	1676	58	3894	4996	1715	59
2 cores 2K, 64B, 4	4852	6278	2180	30	4952	6534	2253	28
2 cores 2K, 128B, 4	4850	6286	2171	30	4962	6544	2280	28
2 cores 2K, 256B, 4	4848	6270	2105	30	4966	6610	2278	28
2 cores 2K, 2K, 4	4800	6228	2176	32	4950	6499	2203	32
2 cores 2K, 16K, 4	4802	6225	2140	30	4952	6523	2279	48
2 cores 2K, 64K, 4	4802	6226	2211	100	5012	6547	2380	102
4 cores 2K, 64B, 4	6926	9099	3101	44	7126	9597	3254	40
4 cores 2K, 128B, 4	6922	9099	3051	44	7146	9596	3230	40
4 cores 2K, 256B, 4	6918	9106	3242	44	7154	9723	3284	40
4 cores 2K, 2K, 4	6822	8985	3054	48	7122	9611	3309	48
4 cores 2K, 16K, 4	6826	8996	3042	76	7126	9589	3253	80
4 cores 2K, 64K, 4	-	-	-	-	-	-	-	-
8 cores 2K, 64B, 4	10037	13308	4478	65	10387	14087	4703	58
8 cores 2K, 128B, 4	10030	13362	4632	65	10422	14162	4833	58
8 cores 2K, 256B, 4	10023	13322	4463	65	10436	14392	4978	58
8 cores 2K, 2K, 4	9855	13175	4475	72	10380	14093	4756	72
8 cores 2K, 16K, 4	-	-	-	-	-	-	-	-
8 cores 2K, 64K, 4	-	-	-	-	-	-	-	-

Tableau A-3 – Résultats de synthèse des architectures avec coeur matériel de sécurité HSC pour les politiques write-through et write-back

impacte sur la pénalité générale du système, puisque les appels au bloc de sécurité sont réduits. Les architectures bénéficient très clairement du couple politique write-back et du coeur de sécurité matériel HSC. De faibles pénalités en latence sont mesurées comparativement aux implémentations sans protection, mais doivent être misent en lumière à l'aide des Tableau A-2 et A-3. Ils font état des surfaces nécessaires pour les architectures proposées, et ne peuvent en aucun cas être dissociés des résultats reportés ici.

A-5 Conclusion

Certes, de nombreux points seraient à détailler et à éclaircir concernant cette annexe. Quoi qu'il en soit, les perspectives que nous laissent présager l'approche multicoeurs, nous montrent que la sécurité n'est pas bornée à un simple microprocesseur. Chose curieuse, peu de publications concernant ce type de sécurité sont à reporter. Il reste donc de la place pour faire des propositions et c'est le but de cette annexe : introduire les contributions de ce manuscrit, pour obtenir une solution robuste.

Bibliographie

Advanced meet-in-the-middle preimage attacks : First results on full tiger, and improved results on MD4 and SHA-2, 2010. URL `http://eprint.iacr.org/2010/016`. A short version of the paper will appear in ASIACRYPT 2010 ntu.guo@gmail.com 14852 received 12 Jan 2010, last revised 30 Aug 2010.

Tiago Alves et Don Felton : TrustZone : Integrated Hardware and Software Security. *ARM White Paper*, juillet 2004.

Jude Angelo Ambrose, Roshan G. Ragel et Sri Parameswaran : RIJID : Random code injection to mask power analysis based side channel attacks. *In Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 489–492, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-627-1. URL `http://doi.acm.org/10.1145/1278480.1278606`.

William Arbaugh, David Farber et Jonathan Smith : A secure and reliable bootstrap architecture. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 65–71, mai 1997.

American Bankers Association : Keyed hash message authentication code, ANSI X9.71, washington, D.C., 2000.

Jean-Philippe Aumasson, Luca Henzen, Willi Meier et Raphael C.-W. Phan : SHA-3 proposal BLAKE. Submission to NIST (Round 1/2), 2008. URL `http://ehash.iaik.tugraz.at/uploads/0/06/Blake.pdf`.

Benoit Badrignans, Reouven Elbaz et Lionel Torres : Secure FPGA Configuration Architecture Preventing System Downgrade. *In Proceedings of the International Conference on Field-Programmable Logic and Applications*, pages 317–322, août 2008.

Elena Gabriela Barrantes, David H. Ackley, Trek S. Palmer, Darko Stefanovic et Dino Dai Zovi : Randomized instruction set emulation to disrupt binary code injection attacks. *In Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 281–289, New York, NY, USA, 2003. ACM. ISBN 1-58113-738-9. URL http://doi.acm.org/10.1145/948109.948147.

Salih Bayar et Arda Yurdakul : Dynamic partial self-reconfiguration on spartan-iii FPGAs via a parallel configuration access port (PCAP).

Juergen Becker, Michael Huebner et Michael Ullmann : Power estimation and power measurement of xilinx virtex FPGAs : Trade-offs and limitations. *In Proceedings of the 16th symposium on Integrated circuits and systems design*, SBCCI '03, pages 283–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2009-X. URL http://dl.acm.org/citation.cfm?id=942808.943944.

Daniel J. Bernstein : Chacha, a variant of salsa20.

Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Sponge Functions. *Ecrypt Hash Workshop 2007*, mai 2007.

Guido Bertoni, Joan Daemen, Michaël Peeters et Gilles Van Assche : Keccak sponge function family main document. Submission to NIST (Round 1), 2008. URL http://keccak.noekeon.org/Keccak-main-1.0.pdf.

Eli Biham, Alex Biryukov et Adi Shamir : Miss in the middle attacks on IDEA and khufu. *In Fast Software Encryption, 6th international Workshop*, pages 124–138. Springer-Verlag, 1999.

Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet et William Jalby : Collisions of SHA-0 and reduced SHA-1. *In* Ronald Cramer, éditeur : *EUROCRYPT*, volume 3494 de *Lecture Notes in Computer Science*, pages 36–57. Springer, 2005. ISBN 3-540-25910-4. URL http://dblp.uni-trier.de/db/conf/eurocrypt/eurocrypt2005.html#BihamCJCLJ05.

Eli Biham et Orr Dunkelman : A framework for iterative hash functions : HAIFA. *In In Proceedings of Second NIST Cryptographic Hash*

Workshop, 2006. URL www.csrc.nist.gov/pki/HashWorkshop/2006/program_2006.htm.

Eli Biham et Adi Shamir : Differential cryptanalysis of DES-like cryptosystems. *In CRYPTO'91*, 1991.

Eli Biham et Adi Shamir : Differential cryptanalysis of the data encryption standard. Springer, 1993.

Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich et Adi Shamir : Key recovery attacks of practical complexity on AES variants with up to 10 rounds.

Alex Biryukov et Dmitry Khovratovich : Related-key cryptanalysis of the full AES-192 and AES-256. *In Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security : Advances in Cryptology*, ASIACRYPT '09, pages 1–18, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-10365-0. URL http://dx.doi.org/10.1007/978-3-642-10366-7_1.

Alex Biryukov, Dmitry Khovratovich et Ivica Nikolić : Distinguisher and related-key attack on the full AES-256. *In Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 231–249, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03355-1. URL http://dl.acm.org/citation.cfm?id=1615970.1615989.

Alex Biryukov et Ivica Nikolić : Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers : Application to AES, Camellia, Khazad and Others. *In* Henri Gilbert, éditeur : *Advances in Cryptology EUROCRYPT 2010*, volume 6110 de *Lecture Notes in Computer Science*, chapitre 17, pages 322–344–344. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13189-9. URL http://dx.doi.org/10.1007/978-3-642-13190-5_17.

Burton H. Bloom : Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13:422–426, July 1970. ISSN 0001-0782. URL http://doi.acm.org/10.1145/362686.362692.

Christophe Bobda, Mateusz Majer, Ali Ahmadinia, Thomas Haller, André Linarth et Jürgen Teich : The erlangen slot machine : Increasing flexibility. *In in FPGA based Reconfigurable Platforms, in Proc. Int'l Conference on Field Programmable Technology (FPT)*, 2005.

Andrey Bogdanov et Christian Rechberger : A 3-subset meet-in-the-middle attack : Cryptanalysis of the lightweight block cipher KTANTAN. *In Proceedings of the 17th international conference on Selected areas in cryptography*, SAC'10, pages 229–240, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-19573-0. URL `http://dl.acm.org/citation.cfm?id=1964441.1964463`.

Pierre Bomel, Guy Gogniat et Jean-Philippe Diguet : A networked, lightweight and partially reconfigurable platform. *In Proceedings of the 4th international workshop on Reconfigurable Computing : Architectures, Tools and Applications*, ARC '08, pages 318–323, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-78609-2. URL `http://dx.doi.org/10.1007/978-3-540-78610-8_35`.

Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid et Yihui Tang : On the false-positive rate of bloom filters. *Inf. Process. Lett.*, 108:210–213, October 2008. ISSN 0020-0190. URL `http://dl.acm.org/citation.cfm?id=1412758.1412983`.

Lawrence Carter et Mark Wegman : Universal classes of hash functions (extended abstract). *In Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC '77, pages 106–112, New York, NY, USA, 1977. ACM. URL `http://doi.acm.org/10.1145/800105.803400`.

Florent Chabaud et Antoine Joux : Differential collisions in SHA-0. *In Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 56–71, London, UK, 1998. Springer-Verlag. ISBN 3-540-64892-5. URL `http://dl.acm.org/citation.cfm?id=646763.706337`.

Cisco ASA 5500 Series Adaptive Security Appliances Models Comparison. Cisco Corporation, 2011. URL `http://www.cisco.com/en/US/products/ps6120/prod_models_comparison.html`.

Jedidiah R. Crandall et Frederic T. Chong : Minos : Control data attack prevention orthogonal to memory model, 2004.

Joan Daemen, Lars Knudsen et Vincent Rijmen : The block cipher SQUARE, 1997.

Joan Daemen et Vincent Rijmen : *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. ISBN 3540425802.

Jean Philippe Delahaye, Guy Gogniat, Christian Roland et Pierre Bomel : Software radio & dynamic reconfiguration on a DSP/FPGA platform. 2004. URL `http://hal.ccsd.cnrs.fr/ccsd-00089395/en/;http://hal.ccsd.cnrs.fr/docs/00/08/93/95/PDF/delahaye_2004frequenz.pdf`.

Sarang Dharmapurikar, Michael Attig et John Lockwood : Design and implementation of a string matching system for network intrusion detection using FPGA-based bloom filters. Rapport technique, Proc. of 12 th Annual IEEE Symposium on FieldProgrammable Custom Computing Machines, 2004a.

Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, John Lockwood et Line Speeds : Deep packet inspection using parallel bloom filters, 2004b.

Kurt Dietrich et Johannes Winter : Secure boot revisited. *In Proceedings of the International Conference for Young Computer Scientists*, pages 2360–2365, novembre 2008.

Saar Drimer : Security for volatile FPGAs. Rapport technique UCAM-CL-TR-763, University of Cambridge, Computer Laboratory, November 2009. URL `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-763.pdf`.

Saar Drimer, Tim Güneysu et Christof Paar : DSPs, BRAMs, and a pinch of logic : Extended recipes for AES on FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 3(1):1–27, 2010. ISSN 1936-7406.

Milenko Drinic et Darko Kirovski : A hardware-software platform for intrusion prevention. *In Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 37, pages 233–242,

Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2126-6. URL http://dx.doi.org/10.1109/MICRO.2004.2.

Orr Dunkelman et Nathan Keller : The effects of the omission of last round's mixcolumns on AES. *Inf. Process. Lett.*, 110:304–308, April 2010. ISSN 0020-0190. URL http://dx.doi.org/10.1016/j.ipl.2010.02.007.

Orr Dunkelman, Nathan Keller et Adi Shamir : Improved single-key attacks on 8-round AES-192 and AES-256. *In* Masayuki Abe, éditeur : *ASIACRYPT*, volume 6477 de *Lecture Notes in Computer Science*, pages 158–176. Springer, 2010. ISBN 978-3-642-17372-1. URL http://dblp.uni-trier.de/db/conf/asiacrypt/asiacrypt2010.html#DunkelmanKS10.

Orr Dunkelman, Gautham Sekar et Bart Preneel : Improved meet-in-the-middle attacks on reduced-round DES. *In Proceedings of the cryptology 8th international conference on Progress in cryptology*, INDOCRYPT'07, pages 86–100, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-77025-9, 978-3-540-77025-1. URL http://dl.acm.org/citation.cfm?id=1777898.1777909.

Adam Dunkels : lwIP. *Computer and Networks Architectures (CNA), Swedish Institute of Computer Science, http ://www.sics.se/ãdam/lwip/*, 2001.

Morris Dworkin : Recommendation for block cipher modes of operation : Galois/counter mode (GCM) and GMAC. *NIST Special Publication 800D-38D*, novembre 2007.

Reouven Elbaz, David Champagne abd Ruby B. Lee, Lionel Torres, Gilles Sassatelli et Pierre Guillemin : TEC-Tree : A low cost and parallelizable tree for efficient defense against memory replay attacks. *In Workshop on Cryptographic Hardware and Embedded Systems*, September 2007.

Reouven Elbaz, Lionel Torres, Gilles Sassatelli, Pierre Guillemin, Michel Bardouillet et Albert Martinez : A parallelized way to provide data encryption and integrity checking on a processor-memory bus. *In Proceedings of the IEEE/ACM International Design Automation Conference*, pages 506–509, July 2006.

ETHERNET : Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer. *IEEE Standard 802.3*.

Li FAN, Pei CAO, Jussara ALMEIDA et Andrei Z. BRODER : Summary cache : a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8:281–293, June 2000. ISSN 1063-6692. URL `http://dx.doi.org/10.1109/90.851975`.

Niels FERGUSON : Authentication weaknesses in GCM. comments submitted to NIST modes of operation process, May 2005.

Niels FERGUSON, John KELSEY, Stefan LUCKS, Bruce SCHNEIER, Michael STAY, David WAGNER et Doug WHITING : Improved cryptanalysis of rijndael. *In Proceedings of the 7th International Workshop on Fast Software Encryption*, FSE '00, pages 213–230, London, UK, 2001. Springer-Verlag. ISBN 3-540-41728-1. URL `http://dl.acm.org/citation.cfm?id=647935.740915`.

Niels FERGUSON, Stefan LUCKS, Bruce SCHNEIER, Doug WHITING, Mihir BELLARE, Tadayoshi KOHNO, Jon CALLAS et Jesse WALKER : The skein hash function family, 2009.

FIPS-140-2 : Security requirements for cryptographic modules. URL `http://www.nist.gov/itl/upload/fips1402.pdf`.

FIPS-180-3 : Secure hash signature standard. URL `http://www.nist.gov/itl/upload/fips180-3_final.pdf`.

FIPS-197 : Advanced encryption standard. URL `http://www.nist.gov/itl/upload/fips-197.pdf`.

FIPS-198-1 : Keyed-hash message authentication code. URL `http://www.nist.gov/itl/upload/FIPS-198-1_final.pdf`.

FIPS-46-3 : Data encryption standard (DES).

Electronic Frontier FOUNDATION : Frequently asked questions (FAQ) about the electronic frontier foundation's DES cracker machine, 1998. URL `http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html`.

Blaise Gassend, Edward Suh, Dwaine Clarke, Marten van Dijk et Srinivas Devadas : Caches and merkle trees for efficient memory integrity verification. *In Proceedings of the International Symposium on High Performance Computer Architecture*, pages 295–306, 2003.

Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer et Søren S. Thomsen : Grøstl – a SHA-3 candidate. Submission to NIST (Round 1/2), 2008. URL `http://groestl.info/Groestl-0.pdf`.

Henri Gilbert et Marine Minier : A collision attack on 7 rounds of rijndael. *In AES Candidate Conference'00*, pages 230–241, 2000.

Jasminder Grewal et John M. DeDourek : Provision of QoS in wireless networks. *In CNSR*, pages 337–340. IEEE Computer Society, 2004. ISBN 0-7695-2096-0. URL `http://csdl.computer.org/comp/proceedings/cnsr/2004/2096/00/20960337abs.htm`.

Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar et Andy Rupp : Cryptanalysis with COPACOBANA. *IEEE Trans. Comput.*, 57: 1498–1513, November 2008. ISSN 0018-9340. URL `http://dl.acm.org/citation.cfm?id=1446228.1446266`.

Kyeong-Soo Han, Kwang-Ok Kim, Tae Whan Yoo et Yul Kwon : The design and implementation of MAC security in EPON. *In Proceedings : International Conference on Advanced Communication Technology*, pages 1676–1680, mai 2006a.

Kyeong-Soo Han, Kwang-Ok Kim, Tae Whan Yoo et Yul Kwon : The design and implementation of MAC security in EPON. *In Proceedings : International Conference on Advanced Communication Technology*, pages 1676–1680, mai 2006b.

Craig Heath et Alexander Klimov : A foundation for secure mobile DRM embedded security. *Wireless Design Magazine*, pages 32–34, décembre 2006.

Christian Henke, Carsten Schmoll et Tanja Zseby : Empirical evaluation of hash functions for multipoint measurements. *SIGCOMM Comput. Commun. Rev.*, 38:39–50, July 2008. ISSN 0146-4833. URL `http://doi.acm.org/10.1145/1384609.1384614`.

Luca Henzen et Wolfgang Fichtner : FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications. *In Proceedings : European Solid State Circuits Conference*, pages 202–205, septembre 2010.

Ching Hu : *Solving Today's Design Security Concerns*. Xilinx Corporation, 2010.

Michael Huebner, Tobias Becker et Juergen Becker : Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguration. *In Proceedings of the 17th symposium on Integrated circuits and system design*, SBCCI '04, pages 28–32, New York, NY, USA, 2004a. ACM. ISBN 1-58113-947-0. URL `http://doi.acm.org/10.1145/1016568.1016583`.

Michael Huebner, Michael Ullmann, Florian Weissel et Juergen Becker : Real-time configuration code decompression for dynamic FPGA self-reconfiguration. *Parallel and Distributed Processing Symposium, International*, 4:138b, 2004b.

Jia Huo, Guochu Shou, Yihong Hu et Zhigang Guo : The design and FPGA implementation of GF(2^{128}) multiplier for GHASH. *In Proceedings : International Conference on Network Security, Wireless Communications and Trusted Computing*, pages 554–557, juin 2009.

IBM : IBM extends enhanced data security to consumer electronics products. URL `http://www-03.ibm.com/press/us/en/pressrelease/19527.wss`.

National Instruments : Building networked applications with the labwindows/CVI UDP support library. January 2009.

Intel : Execute disable bit and enterprise security. URL `http://www.intel.com/technology/xdbit/index.htm`.

Intel : Thunderbolt™ technology, 2011.

Takanori Isobe : A single-key attack on the full GOST block cipher. *In Proceedings of the 18th international conference on Fast software encryption*, FSE'11, pages 290–305, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21701-2. URL `http://dl.acm.org/citation.cfm?id=2022159.2022183`.

Don Johnson, Alfred Menezes et Scott A. Vanstone : The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Sec.*, pages 36–63, 2001.

Adam Kirsch et Michael Mitzenmacher : Less hashing, same performance : Building a better bloom filter. *Random Struct. Algorithms*, 33:187–218, September 2008. ISSN 1042-9832. URL `http://dl.acm.org/citation.cfm?id=1400123.1400125`.

Adam Kirsch, Michael Mitzenmacher et George Varghese : Algorithms for next generation networks, computer communications and networks, February 2010.

Dirk Koch, Christian Beckhoff et Jim Torresen : Demo paper : Advanced partial run-time reconfiguration on spartan-6 FPGAs. *In Proceedings of International Conference on Field-Programmable Technology (ICFPT́10)*, Beijing, China, 2010. IEEE. to apear.

Hugo Krawczyk, Mihir Bellare et Ran Canetti : HMAC : Keyed-hashing for message authentication. RFC 2104 (Informational), February 1997a. URL `http://www.ietf.org/rfc/rfc2104.txt`.

Hugo Krawczyk, Mihir Bellare et Ran Canetti : Hmac : Keyed-hashing for message authentication, internet engineering task force, request for comments (RFC) 2104, February 1997b.

Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer et Manfred Schimmler : Breaking ciphers with COPACOBANA – a cost-optimized parallel code breaker. *In IN WORKSHOP ON CRYPTOGRAPHIC HARDWARE and EMBEDDED SYSTEMS - CHES 2006,YOKOHAMA*, pages 101–118. Springer Verlag, 2006.

Ian Kuon, Student Member, Jonathan Rose et Senior Member : Measuring the gap between FPGAs and ASICs. *In In Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA'06*, pages 21–30. ACM Press, 2006.

Jean LaBrosse : *MicroC/OS-II : The Real-Time Kernel*. CMP Books, San Francisco, CA, 2002.

Arnaud Lagger, Andres Upegui, Eduardo Sanchez et Ivan Gonzalez : Self-reconfigurable pervasive platform for cryptographic application.

Field Programmable Logic and Applications, 2006. FPL '06. International Conference on, pages 1–4, Aug. 2006.

Pierre L'Ecuyer et Richard Simard : TestU01 : A C Library for Empirical Testing of Random Number generators. *ACM Transactions on Mathematical Software*, 33(4):22 :1–22 :40, August 2007.

Kwangyoon Lee et Alex Orailoglu : Application specific non-volatile primary memory for embedded systems. *In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 31–36, 2008.

Manhee Lee, Minseon Ahn et Eun Jung Kim : I2SEMS : Interconnects-independent security enhanced shared memory multiprocessor systems. *In Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, PACT '07, pages 94–103, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2944-5. URL `http://dx.doi.org/10.1109/PACT.2007.39`.

David Lie, Chandramohan Thekkath et Mark Horowitz : Implementing an untrusted operating system on trusted hardware. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 178–192, October 2003.

David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell et Mark Horowitz : Architectural support for copy and tamper resistant software. *In Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 168–177, 2000.

Moses Liskov, Ronald L. Rivest et David Wagner : Tweakable block ciphers. *In Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '02, pages 31–46, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44050-X. URL `http://dl.acm.org/citation.cfm?id=646767.704290`.

Chenghuai Lu, Tao Zhang, Weidong Shi et Hsien-Hsin S. Lee : M-TREE : a high efficiency security architecture for protecting integrity and privacy of software. *J. Parallel Distrib. Comput.*, 66:1116–1128, September 2006.

ISSN 0743-7315. URL http://dx.doi.org/10.1016/j.jpdc.2006.04.011.

Yang Lu, Guochu Shou, Yihong Hu et Zhigang Guo : The research and efficient FPGA implementation of Ghash core for GMAC. *In Proceedings : International Conference on E-Business and Information System Security*, pages 1–5, mai 2009.

Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen et Mahmoud Modarres-Hashemi : Improved impossible differential cryptanalysis of 7-round AES-128. *In* Guang Gong et Kishan Chand Gupta, éditeurs : *INDOCRYPT*, volume 6498 de *Lecture Notes in Computer Science*, pages 282–291. Springer, 2010. ISBN 978-3-642-17400-1. URL http://dblp.uni-trier.de/db/conf/indocrypt/indocrypt2010.html#MalaDRM10.

George Marsaglia : The marsaglia random number CDROM including the diehard battery of tests of randomness, 1995.

George Marsaglia et Wai Wan Tsang : Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):1–9, 1 2002. ISSN 1548-7660. URL http://www.jstatsoft.org/v07/i03.

Mitsuru Matsui : Linear cryptanalysis method for DES cipher. *In Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '93, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc. ISBN 3-540-57600-2. URL http://dl.acm.org/citation.cfm?id=188307.188366.

Makoto Matsumoto et Takuji Nishimura : Mersenne twister : a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

MAXIM : Increasing system security by using the DS5250 as a secure coprocessor. URL http://www.maxim-ic.com/appnotes.cfm/appnote_number/3294.

David McGrew et John Viega : *The Galois/Counter Mode of Operation (GCM)*, 2004a. Submission to NIST Modes of Operation Process.

David McGrew et John Viega : The security and performance of the galois/counter mode (GCM) of operation. *In In INDOCRYPT, volume 3348 of LNCS*, pages 343–355. Springer, 2004b.

David McGrew et John Viega : The galois/counter mode of operation (GCM). updated submission to NIST, modes of operation process, 2005.

David McGrew et John Viega : The use of galois message authentication code (GMAC) in ipsec ESP and AH. RFC 4543 (Proposed Standard), mai 2006. URL http://www.ietf.org/rfc/rfc4543.txt.

Aleksandar Milenkovic, Milena Milenkovic et Emil Jovanov : An efficient runtime instruction block verification for secure embedded systems. *J. Embedded Comput.*, 2:57–76, January 2006. ISSN 1740-4460. URL http://dl.acm.org/citation.cfm?id=1370986.1370992.

Milena Milenkovic : *Architectures for Run-time Verification of Code Integrity*. Thèse de doctorat, Huntsville, AL, USA, 2005. AAI3164054.

Milena Milenković, Aleksandar Milenković et Emil Jovanov : Hardware support for code integrity in embedded processors. *In Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, CASES '05, pages 55–65, New York, NY, USA, 2005a. ACM. ISBN 1-59593-149-X. URL http://doi.acm.org/10.1145/1086297.1086306.

Milena Milenković, Aleksandar Milenković et Emil Jovanov : Using instruction block signatures to counter code injection attacks. *SIGARCH Comput. Archit. News*, 33:108–117, March 2005b. ISSN 0163-5964. URL http://doi.acm.org/10.1145/1055626.1055641.

Michael Mitzenmacher : Compressed bloom filters. *In Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, PODC '01, pages 144–150, New York, NY, USA, 2001. ACM. ISBN 1-58113-383-9. URL http://doi.acm.org/10.1145/383962.384004.

Michael Mitzenmacher et Salil Vadhan : Why simple hash functions work : Exploiting the entropy in a data stream. *In Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 746–755, Philadelphia, PA, USA, 2008. Society for Industrial

and Applied Mathematics. URL http://dl.acm.org/citation.cfm?id=1347082.1347164.

Recommendation for Block Cipher Modes of Operation : The CCM Mode for Authentication and Confidentiality. National Institute of Standards and Technology, 2004. Special publication 800-38C.

Recommendation for Block Cipher Modes of Operation : Galois/Counter Mode (GCM) and (GMAC). National Institute of Standards and Technology, 2007. Special publication 800-38D.

Anna Ostlin et Rasmus Pagh : Uniform hashing in constant time and linear space. *In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 622–628, New York, NY, USA, 2003. ACM. ISBN 1-58113-674-9. URL http://doi.acm.org/10.1145/780542.780633.

Hilmi Ozdoganoglu, T. N. Vijaykumar, Carla E. Brodley, Benjamin A. Kuperman et Ankit Jalote : Smashguard : A hardware solution to prevent security attacks on the function return address. *IEEE Trans. Comput.*, 55:1271–1285, October 2006. ISSN 0018-9340. URL http://dl.acm.org/citation.cfm?id=1159156.1159226.

Christof Paar : Implementation options for finite field arithmetic for elliptic curve cryptosystems. *In Proceedings : Elliptic Curve Cryptosystems Workshop*, novembre 1999.

Christof Paar et Jan Pelzl : *Understanding Cryptography : A Textbook for Students and Practitioners*. Springer-Verlag New York Inc, 2010.

Anna Pagh, Rasmus Pagh et S. Srinivasa Rao : An optimal bloom filter replacement. *In Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 823–829, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. ISBN 0-89871-585-7. URL http://dl.acm.org/citation.cfm?id=1070432.1070548.

Marco Pasotti, Guido De Sandre, David Iezzi, Davide Lena, Gilberto Muzzi, Marco Poles et Pier Luigi Rolandi : An application specific embeddable flash memory system for non-volatile storage of code, data and bit-streams for embedded FPGA configurations. *In Proceedings of the Symposium on VLSI Circuits*, pages 213–216, juin 2003.

Krutartha Patel, Sridevan Parameswaran et Seng Lin Shee : Ensuring secure program execution in multiprocessor embedded systems : a case study. *In Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '07, pages 57–62, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-824-4. URL http://doi.acm.org/10.1145/1289816.1289833.

G. Perrotey et P. Bressy : Embedded devices : Security implementation. URL http://www.secure-machines.com/fichiers/technical%20white%20Paper-Fev2006.pdf.

Jürg Platte et Edwin Naroska : A combined hardware and software architecture for secure computing. *In Proceedings of the 2nd conference on Computing frontiers*, CF '05, pages 280–288, New York, NY, USA, 2005. ACM. ISBN 1-59593-019-1. URL http://doi.acm.org/10.1145/1062261.1062308.

Nicholas J. Ploplys et Andrew G. Alleyne : UDP network communications for distributed wireless control. *American Control Conference, 2003. Proceedings of the 2003*, 4:3335–3340 vol.4, June 2003. ISSN 0743-1619.

Ely Porat : An optimal bloom filter replacement based on matrix solving. *In Proceedings of the Fourth International Computer Science Symposium in Russia on Computer Science - Theory and Applications*, CSR '09, pages 263–273, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03350-6. URL http://dx.doi.org/10.1007/978-3-642-03351-3_25.

Roshan G. Ragel et Sri Parameswaran : IMPRES : Integrated monitoring for processor reliability and security. *In Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 502–505, New York, NY, USA, 2006. ACM. ISBN 1-59593-381-6. URL http://doi.acm.org/10.1145/1146909.1147041.

RAKNET : Raknet www.jenkinssoftware.com.

Karthick Ramu et Chethan Ananth : Fully pipelined AES with speed exceeding 20 gbps using logic-only implementation of s-box. URL teal.gmu.edu/courses/ECE746/project/slides_2008/AES_pipelined_slides.pdf.

Srivaths Ravi, Anand Raghunathan et Srimat Chakradhar : Tamper resistance mechanisms for secure, embedded systems. *In Proceedings of the 17th International Conference on VLSI Design*, VLSID '04, pages 605–, Washington, DC, USA, 2004a. IEEE Computer Society. ISBN 0-7695-2072-3. URL http://dl.acm.org/citation.cfm?id=962758.963491.

Srivaths Ravi, Anand Raghunathan, Paul Kocher et Sunil Hattangady : Security in embedded systems : Design challenges. *ACM Trans. Embed. Comput. Syst.*, 3:461–491, August 2004b. ISSN 1539-9087. URL http://doi.acm.org/10.1145/1015047.1015049.

Internet Engineering Task Force. RFC1122 : Requirements for internet hosts – communication layers. October 1989.

Internet Engineering Task Force. RFC768 : User datagram protocol. August 1980.

Internet Engineering Task Force. RFC793 : Transmission control protocol. September 1981.

Abdul R. Rind, Shahzad Khurram et M. Abdul Qadir : Evaluation and comparison of TCP and UDP over wired-cum-wireless LAN. *Multitopic Conference, 2006. INMIC '06. IEEE*, pages 337–342, Dec. 2006.

Austin Rogers : *Designing Cost-effective Secure Processors for Embedded Systems : Principles, Challenges, and Architectural Solutions*. Thèse de doctorat, Huntsville, AL, USA, 2010. AAI3410781.

Brian Rogers, Siddhartha Chhabra, Milos Prvulovic et Yan Solihin : Using address independent seed encryption and bonsai merkle trees to make secure processors OS- and performance-friendly. *In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 183–196, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3047-8. URL http://dx.doi.org/10.1109/MICRO.2007.44.

Brian Rogers, Chenyu Yan, Siddhartha Chhabra, Milos Prvulovic et Yan Solihin : Single-level integrity and confidentiality protection for distributed shared memory multiprocessors. *In HPCA*, pages 161–172. IEEE Computer Society, 2008. URL http://dblp.uni-trier.de/db/conf/hpca/hpca2008.html#RogersYCPS08.

Christian Esteve Rothenberg, Carlos Alberto Braz Macapuna, Fábio Luciano Verdi et Maurício F. Magalhães : The deletable bloom filter : A new member of the bloom family. *CoRR*, abs/1005.0352, 2010.

Markku-Juhani O. Saarinen : Finding GCM weak keys, 2011a.

Markku-Juhani O. Saarinen : GCM, GHASH and Weak keys. Cryptology ePrint Archive, Report 2011/202, 2011b. `http://eprint.iacr.org/`.

Yu Sasaki et Kazumaro Aoki : Finding preimages in full MD5 faster than exhaustive search. *In Proceedings of the 28th Annual International Conference on Advances in Cryptology : the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 134–152, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-01000-2. URL `http://dx.doi.org/10.1007/978-3-642-01001-9_8`.

SATA-IO : New SATA spec will double data transfer rates to 6 gbit/s, 2009.

Weidong Shi, Hsien-Hsin S. Lee, Mrinmoy Ghosh et Chenghuai Lu : Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems. *In Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, PACT '04, pages 123–134, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2229-7. URL `http://dx.doi.org/10.1109/PACT.2004.8`.

Victor Shoup : On fast and provably secure message authentication based on universal hashing. *In Proceedings : Advances in Cryptology*, pages 313–328, août 1996.

Haoyu Song, Sarang Dharmapurikar, Jonathan Turner et John Lockwood : Fast hash table lookup using extended bloom filter : an aid to network processing. *SIGCOMM Comput. Commun. Rev.*, 35:181–192, August 2005. ISSN 0146-4833. URL `http://doi.acm.org/10.1145/1090191.1080114`.

Junhyuk Song, Radha Poovendran, Jicheol Lee et Tetsu Iwata : The AES-CMAC Algorithm. RFC 4493 (Informational), juin 2006. URL `http://www.ietf.org/rfc/rfc4493.txt`.

Marc Stevens, Arjen Lenstra et Benne Weger : Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. *In Proceedings of the 26th annual international conference on Advances in Cryptology*, EUROCRYPT '07, pages 1–22, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72539-8. URL http://dx.doi.org/10.1007/978-3-540-72540-4_1.

Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik et Benne Weger : Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. *In Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 55–69, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03355-1. URL http://dl.acm.org/citation.cfm?id=1615970.1615976.

Chih-Pin Su, Chen-Hsing Wang, Kuo-Liang Cheng, Chih-Tsun Huang et Cheng-Wen Wu : Design and test of a scalable security processor. *In Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 372–375, New York, NY, USA, 2005. ACM. ISBN 0-7803-8737-6. URL http://doi.acm.org/10.1145/1120725.1120872.

Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk et Srinivas Devadas : AEGIS : Architecture for tamper-evident and tamper-resistant processing. *In Proceedings of the International Conference on Supercomputing*, pages 160–171, 2003a.

Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk et Srinivas Devadas : Efficient memory integrity verification and encryption for secure processors. *In Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 339–350, 2003b.

Edward Suh, Jae W. Lee, David Zhang et Srinivas Devadas : Secure 0rogram execution via dynamic information flow tracking. *In Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XI, pages 85–96, New York, NY, USA, 2004. ACM. ISBN 1-58113-804-0. URL http://doi.acm.org/10.1145/1024393.1024404.

Edward Suh, Charles W. O'Donnell, Ishan Sachdev et Srinivas Devadas : Design and implementation of the AEGIS single-chip secure processor using physical random functions. *In Proceedings of the International Symposium on Computer Architecture*, pages 25–36, 2005.

Sasu Tarkoma, Christian Esteve Rothenberg et Eemil Lagerspetz : Theory and practice of bloom filters for distributed systems. Second issue 2012.

Nathan Tuck, Brad Calder et George Varghese : Hardware and binary modification support for code pointer protection from buffer overflow. *In Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 37, pages 209–220, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2126-6. URL `http://dx.doi.org/10.1109/MICRO.2004.20`.

Tomohisa Uchida : Hardware-based TCP processor for gigabit ethernet. *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, 1: 309–315, 26 2007-Nov. 3 2007. ISSN 1082-3654.

George Varghese : Network algorithmics : An interdisciplinary approach to designing fast networked devices (the morgan kaufmann series in networking), 2004.

Romain Vaslin, Guy Gogniat, Jean-Philippe Diguet, Russell Tessier, Deepak Unnikrishnan et Kris Gaj : Memory security management for reconfigurable rmbedded systems. *In Proceedings of the IEEE Conference on Field Programmable Technology*, pages 153–160, décembre 2008.

Jimei Wang, Guochu Shou, Yihong Hu et Zhigang Guo : High-speed architectures for GHASH based on efficient bit-parallel multipliers. *In Proceedings : IEEE International Conference on Wireless Communications, Networking and Information Security*, pages 582–586, juin 2010.

Xiaoyun Wang, Yiqun Lisa Yin et Hongbo Yu : Finding collisions in the full SHA-1. *In In Proceedings of Crypto*, pages 17–36. Springer, 2005.

Xiaoyun Wang et Hongbo Yu : How to break MD5 and other hash functions. *In In EUROCRYPT*. Springer-Verlag, 2005.

Zhenghong Wang et Ruby B. Lee : New cache designs for thwarting software cache-based side channel attacks. *In Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 494–505, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-706-3. URL http://doi.acm.org/10.1145/1250662.1250723.

Mark Wegman et Lawrence Carter : New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, juin 1981.

Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang et San Ling : Improved meet-in-the-middle cryptanalysis of KTANTAN. *In Proceedings of the 16th Australasian conference on Information security and privacy*, ACISP'11, pages 433–438, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22496-6. URL http://dl.acm.org/citation.cfm?id=2029853.2029891.

Kyu-Young Whang, Brad T. Vander-Zanden et Howard M. Taylor : A linear-time probabilistic counting algorithm for database applications. *ACM Trans. Database Syst.*, 15:208–229, June 1990. ISSN 0362-5915. URL http://doi.acm.org/10.1145/78922.78925.

WIFI : Wireless LAN medium access control (MAC) and physical layer (PHY) specifications.

John W. Williams et Neil Bergmann : Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. *In* Toomas P. Plaks, éditeur : *ERSA*, pages 163–169. CSREA Press, 2004. ISBN 1-932415-42-4.

Hongjun Wu : The hash function JH. Submission to NIST (Round 1/2), 2009. URL http://ehash.iaik.tugraz.at/uploads/1/1d/Jh20090915.pdf.

Xilinx : Xapp433. web server design using microblaze soft processor. October 2006.

Lock Your Designs with the Virtex-4 Security Solution. Xilinx Corporation, 2005.

Microblaze Processor Reference Guide. Xilinx Corporation, 2009.

Spartan-6 Family Overview. Xilinx Corporation - DS160, March 2010a.

SP605 Hardware User Guide. Xilinx Corporation - UG526, June 2010b.

Jun Xu, Zbigniew Kalbarczyk, Sanjay Patel et Ravishankar K. Iyer : Architecture support for defending against buffer overflow attacks. 2002.

Chenyu Yan, Brian Rogers, Daniel Englender, Yan Solihin et Milos Prvulovic : Improving cost, performance, and security of memory encryption and authentication. *In Proceedings of the International Symposium on Computer Architecture*, pages 179–190, juillet 2006.

Jun Yang, Lan Gao et Youtao Zhang : Improving memory encryption performance in secure processors. *IEEE Trans. Comput.*, 54:630–640, May 2005. ISSN 0018-9340. URL `http://dx.doi.org/10.1109/TC.2005.80`.

Joseph Zambreno, Alok Choudhary, Rahul Simha, Bhagi Narahari et Nasir Memon : SAFE-OPS : An approach to embedded software security. *ACM Trans. Embed. Comput. Syst.*, 4:189–210, February 2005. ISSN 1539-9087. URL `http://doi.acm.org/10.1145/1053271.1053279`.

Alan Zeichick : Security ahoy ! flying the NX flag on windows and AMD64 to stop attacks. URL `http://developer.amd.com/articlex.jsp?id=143`.

Youtao Zhang, Lan Gao, Xiangyu Zhang et Rajiv Gupta : SENSS : Security enhancement to symmetric shared memory multiprocessors. *In In Intl. Symp. on High-Performance Computer Architecture*, pages 352–362. IEEE Computer Society, 2005.

Gang Zhou, Harald Michalik et László Hinsenkamp : Improving throughput of AES-GCM with pipelined Karatsuba multipliers on FPGAs. *In Proceedings : International Workshop on Reconfigurable Computing : Architectures, Tools and Applications*, pages 193–203, mars 2009.