N° d'ordre : 4388

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DES SCIENCES PHYSIQUES DE L'INGÉNIEUR

Par Michaël GRAND

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : ÉLECTRONIQUE

Conception d'un crypto-système reconfigurable pour la radio logicielle sécurisée

Soutenue le : 02 Décembre 2011

Devant la commission d'examen formée de :

M. M. Auguin	Directeur de recherche, LEAT, UNS, Valbonne	Rapporteur
M. J.-L. Danger	Directeur d'étude, Telcom ParisTech, Paris	Rapporteur
M. D. Dallet	Professeur, Lab. IMS, IPB, Bordeaux	Directeur de thèse
M. G. Gogniat	Professeur, LESTER, UBS, Lorient	Co-Directeur de thèse
M. L. Bossuet	Maître de conférence, Lab. H.-C., Telecom St. Etienne	Examinateur
M. C. Jego	Professeur, Lab. IMS, IPB, Bordeaux	Examinateur
M. J.-P. Delahaye	Ingénieur, DGA Maîtrise de l'information, Bruz	Examinateur

Résumé

Les travaux de recherche détaillés dans ce document portent sur la conception et l'implantation d'un composant matériel jouant le rôle du sous-système cryptographique d'une radio logicielle sécurisée.

A partir du début des années 90, les systèmes radios ont peu à peu évolué de la radio classique vers la radio logicielle. Le développement de la radio logicielle a permis l'intégration d'un nombre toujours plus grand de standards de communication sur une même plateforme matérielle. La réalisation concrète d'une *radio logicielle sécurisée* amène son concepteur à faire face à de nombreuses problématiques qui peuvent se résumer par la question suivante : Comment implanter un maximum de standards de communication sur une même plateforme matérielle et logicielle ? Ce document s'intéresse plus particulièrement à l'implantation des *standards cryptographiques* destinés à protéger les radiocommunications.

Idéalement, la solution apportée à ce problème repose exclusivement sur l'utilisation de processeurs numériques. Cependant, les algorithmes cryptographiques nécessitent le plus souvent une puissance de calcul telle que leur implantation sous forme logicielle n'est pas envisageable. Il s'ensuit qu'une radio logicielle doit parfois intégrer des composants matériels dédiés dont l'utilisation entre en conflit avec la propriété de flexibilité propre aux radios logicielles.

Or depuis quelques années, le développement de la technologie FPGA a changé la donne. En effet, les derniers FPGA embarquent un nombre de ressources logiques suffisant à l'implantation des fonctions numériques complexes utilisées par la radio logicielle. Plus précisément, la possibilité offerte par les FPGA d'être reconfigurés dans leur totalité (voir même partiellement pour les derniers d'entre eux) fait d'eux des candidats idéaux à l'implantation de composants matériels flexibles et évolutifs dans le temps.

À la suite de ces constatations, des travaux de recherche ont été menés au sein de l'équipe *Conception des Systèmes Numériques* du Laboratoire IMS. Ces travaux ont d'abord débouché sur la publication d'une architecture de *sous-système cryptographique* pour la *radio logicielle sécurisée* telle qu'elle est définie par la *Software Communication Architecture*. Puis, ils se sont poursuivis par la conception et l'implantation d'un *cryptoprocesseur multicœur dynamiquement reconfigurable* sur FPGA.

Mots clés : radio logicielle sécurisée, *Software Communication Architecture*, sous-système cryptographique, cryptoprocesseur multicœur, FPGA, reconfiguration dynamique partielle.

Abstract

The research detailed in this document deal with the design and implementation of a hardware integrated circuit intended to be used as a *cryptographic sub-system* in *secure software defined radios*.

Since the early 90's, radio systems have gradually evolved from traditional radio to *software defined radio*. Improvement of the *software defined radio* has enabled the integration of an increasing number of communication standards on a single radio device. The designer of a *software defined radio* faces many problems that can be summarized by the following question: How to implement a maximum of communication standards into a single radio device? Specifically, this work focuses on the implementation of *cryptographic standards* aimed to protect radio communications.

Ideally, the solution to this problem is based exclusively on the use of digital processors. However, cryptographic algorithms usually require a large amount of computing power which makes their software implementation inefficient. Therefore, a *secure software defined radio* needs to incorporate dedicated hardware even if this usage is conflicting with the property of flexibility specific to *software defined radios*.

Yet, in recent years, the improvement of FPGA circuits has changed the deal. Indeed, the latest FPGAs embed a number of logic gates which is sufficient to meet the needs of the complex digital functions used by *software defined radios*. The possibility offered by FPGAs to be reconfigured in their entirety (or even partially for the last of them) makes them ideal candidates for implementation of hardware components which have to be flexible and scalable over time.

Following these observations, research was conducted within the *Conception des Systèmes Numériques* team of the IMS laboratory. These works led first to the publication of an architecture of *cryptographic subsystem* compliant with the *security supplement* of the *Software Communication Architecture*. Then, they continued with the design and implementation of a *partially reconfigurable multi-core cryptoprocessor* intended to be used in the latest FPGAs.

Key words : Secure Software Defined Radio, Software Communication Architecture, Cryptographic Sub System, Multi-core Cryptoprocessor, FPGA, partial reconfiguration.

Remerciements

Ce manuscrit retrace de façon synthétique l'ensemble des travaux de recherche que j'ai développés durant ces trois dernières années en tant que doctorant au laboratoire IMS (Intégration du Matériau au Système) de Bordeaux. Je tiens tout d'abord à remercier le personnel du laboratoire IMS, de la DGA et du CNRS qui m'ont fourni le soutient matériel et financier sans lesquels je n'aurais pu mener à bien ces travaux.

Je remercie Michael Auguin et Jean-Luc Danger qui ont accepté d'être les rapporteurs de ma thèse. Leurs remarques pertinentes ont sans nul doute permis d'améliorer la qualité de mes réflexions. Je remercie aussi Jean-Philippe Delahaye qui en tant que représentant de la DGA a pu apporter un avis d'industriel sur mes travaux notamment en ce qui concerne la radio logicielle. Pour finir, je remercie Christophe Jego pour sa participation à ma soutenance de thèse en tant que président de jury.

Je remercie particulièrement Lilian Bossuet et Guy Gogniat, mes deux encadrants de thèse, qui ont su m'aider à garder le cap dans la jungle des possibilités de travaux qui s'offraient à moi. Leurs lectures attentives, à l'espace prêt, ainsi que leurs remarques constructives m'ont permis de grandement améliorer la qualité de mes publications et de ce manuscrit. Je remercie aussi Dominique Dallet, mon directeur de thèse, qui m'a aidé à améliorer la pédagogie et la structuration de mes publications. Je remercie enfin Bertrand Le Gal qui m'a amené, au cours de débats acharnés, à me poser les bonnes questions.

Je remercie mes amis doctorants, ingénieurs et stagiaires, avec qui j'ai passé ces trois dernières années. Aurélien, Moez, Sahbi et Simon ont su égayer les longues journées que j'ai passées à rédiger mes articles et mon manuscrit de thèse. Je remercie aussi Romain Sacheau, mon stagiaire en troisième année d'école d'ingénieur, qui m'a permis de découvrir les arcanes du flot de reconfiguration partielle de Xilinx.

Pour finir, je remercie chaleureusement les nombreux participants de la « Liste du midi » (et notamment les filles) qui ont réussi à supporter mon humour chaque midi pendant trois ans. Merci à vous tous !

Table des matières

Liste des figures

Liste des tables

Introduction

Du simple tag RFID inséré dans l'étiquette d'un livre au système de téléphonie par satellite, il faut admettre que les composants et terminaux électroniques sans fil font aujourd'hui partie de notre quotidien. En 2009, on comptabilisait en France un peu plus de 59 millions de souscriptions aux services de téléphonie mobile. Et, selon l'institut de mesure d'audience Médiamétrie-eStat[1], 17,7 millions d'utilisateurs se sont connectés à l'internet mobile au deuxième trimestre 2011. Ces deux chiffres illustrent clairement la monté en puissance des communications sans fils et des terminaux mobiles. Sans s'attarder sur les raisons de cet engouement, on constate que le principal facteur ayant rendu cette évolution possible réside dans l'amélioration sans commune mesure des composants électroniques. Concrètement, ces améliorations se traduisent par une diminution des coûts de fabrication, une miniaturisation toujours plus poussée et des performances (consommation, puissance de calcul) sans cesse améliorées.

Parallèlement à la démocratisation des terminaux sans fils, il s'est produit leur convergence vers un système unique à usage multiple. Aujourd'hui, un « smartphone » est tout à la fois un téléphone mobile servant de point d'accès à internet (3G), un baladeur numérique sans fil (Bluetooth), un outil de positionnement (GPS) et un outils de paiement (NFC[2]). Chacun de ses usages est basé sur l'utilisation de composants électroniques sans fils. Or, rien de tout cela ne serait possible si les technologies de communication sans fil n'avaient pas évolué progressivement d'une construction exclusivement analogique vers une construction principalement numérique. Dans les faits, le passage au numérique fut l'un des facteurs clé à l'origine de la miniaturisation et de la baisse des coûts de fabrication des terminaux. En effet, contrairement aux composants analogiques, nombreux sont les composants numériques à être configurables voir programmables. Cette caractéristique a peu à peu permis d'intégrer sur une même puce de nombreuses fonctionnalités différentes. Ce sont des composants numériques tels que les DSP qui ont permis l'émergence de la *radio numérique* qui se trouve être la condition sine qua non à l'apparition des terminaux mobiles tels que nous les connaissons aujourd'hui.

La majorité des nouveaux standards de communication sans fil sont aujourd'hui basés sur l'utilisation de la *radio numérique* et des *communications numériques*. Or, le nombre toujours plus grand de standards de communication (ex. GSM, WIFI, DVB, etc.) allié à la convergence des moyens de communication font qu'il est devenu nécessaire de développer des systèmes de communications multistandards. Ces systèmes sont majoritairement basés sur le paradigme de la *radio logicielle*. Historiquement, le terme de *radio logicielle* est apparu dès 1984

[1] http://www.mediametrie.fr
[2] Near Field Communication

dans les bureaux de la société E-Systems qui travaillait alors avec le gouvernement américain à la fabrication d'un système de communication radio multistandard destiné à l'armée. L'intérêt de la radio logicielle peut se résumer de la façon suivante : « obtenir d'un même système matériel plusieurs fonctions radios différentes ». Par analogie, on pourrait comparer cela au fait qu'un même ordinateur personnel permet l'exécution de différentes applications contrairement à une simple calculette .

Pour arriver à un tel résultat, il est nécessaire de positionner les interfaces de conversion analogique/numérique aussi près que possible des antennes afin de tirer parti de la flexibilité des systèmes numériques. Il s'ensuit que : plus les composants numériques utilisés dans une radio logicielle sont performants, plus cette radio est performante. Selon les besoins et contraintes de performances relatifs à la conception d'une radio logicielle, différentes solutions sont envisageables : GPP, DSP, FPGA, ASIC ou encore une combinaison de ceux-ci. SPEAKeasy [CoBo99] développée au cours des années 90 fut la première radio logicielle militaire mise au point par le gouvernement américain. Elle était effectivement basée sur l'utilisation d'une combinaison de ces composants et, fait remarquable à l'époque, utilisait pour la première fois la technologie FPGA pour traiter les signaux radio.

Bien sûr, le besoin en termes de flexibilité et performance de la *radio logicielle* ne se réduit pas à sa partie matérielle. La couche logicielle d'une telle radio doit aussi faire preuve de flexibilité en étant *portable*. Il s'avère que le développement des couches logicielles destinées aux radios logicielles est un processus long et fastidieux qui, lorsqu'il n'est pas standardisé, conduit à la conception de logiciels ayant une *réutilisabilité* et une *portabilité* quasi nulle. Afin de répondre à cette problématique, le gouvernement américain publia au début des années 2000 les spécifications d'une plateforme logicielle destinée à favoriser l'interopérabilité et la réutilisabilité des composants logiciels des radios compatibles. Ce standard baptisé *Software Communication Architecture* (SCA) [Jtrs10] a subi de nombreuses évolutions au cours de ces dix dernières années. Car, bien que destinée entre autres aux applications militaires, la question de la sécurité des radios logicielles n'y est jamais abordée. Pour répondre à ce problème, le gouvernement américain publia en 2004 un supplément sécurité à la SCA [Jtrs04] qui spécifie et définit les différents mécanismes à mettre en œuvre pour sécuriser une radio logicielle et les flux d'informations qui la traverse. Malheureusement, ces spécifications logicielles n'ont jamais été développées en prenant en compte les limitations des composants numériques spécialisés. En réalité, celles-ci ont été principalement conçues dans l'optique d'une implantation logicielle sur processeur généraliste. Or, de nombreux calculs mathématiques, notamment ceux issus de la cryptographie, requièrent une puissance de calcul qui ne peut être fournie que par des composants spécialisés du type ASIC ou FPGA.

Manquant de maturité, les composants FPGA n'ont que tardivement été pris en compte par les spécifications de la SCA avec, dans un premier temps, la publication d'un supplément à la SCA baptisé MHAL pour *Modem Hardware Abstraction Layer* puis, dans un second temps, une intégration directe au standard SCA dans sa dernière version. Cette

montée en puissance de l'intégration des FPGA aux standards de la radio logicielle n'est pas étonnante lorsqu'on s'attarde sur l'évolution de cette technologie. Le premier FPGA commercialisé en 1984 par la société Xilinx, le XCV2000, avait une capacité de 1500 portes logiques. Aujourd'hui, les derniers FPGA de chez Xilinx ont des capacités de plusieurs millions de portes logiques. A titre d'exemple, le plus gros FPGA SRAM Virtex 6 de Xilinx embarque 474240 LUT à 6 entrées, 948480 Flip-Flop et 26 Mb de SRAM. Si l'on ajoute à cela, les blocs DSP, les PLL, les ports d'entrées/sorties haut débit ou encore la compatibilité avec le PCI-Express, on devine ici l'importance du bond technologique. De plus, tandis que le coût de fabrication par unité d'un ASIC ne fait qu'augmenter, celui des FPGA ne fait que diminuer, comme l'illustre la Figure 1 issus des travaux de N. Tredennick [TrSh03]. Il s'ensuit qu'au fil du temps les solutions à base de FPGA sont devenues de plus en plus intéressantes pour la production en petite et moyenne série.

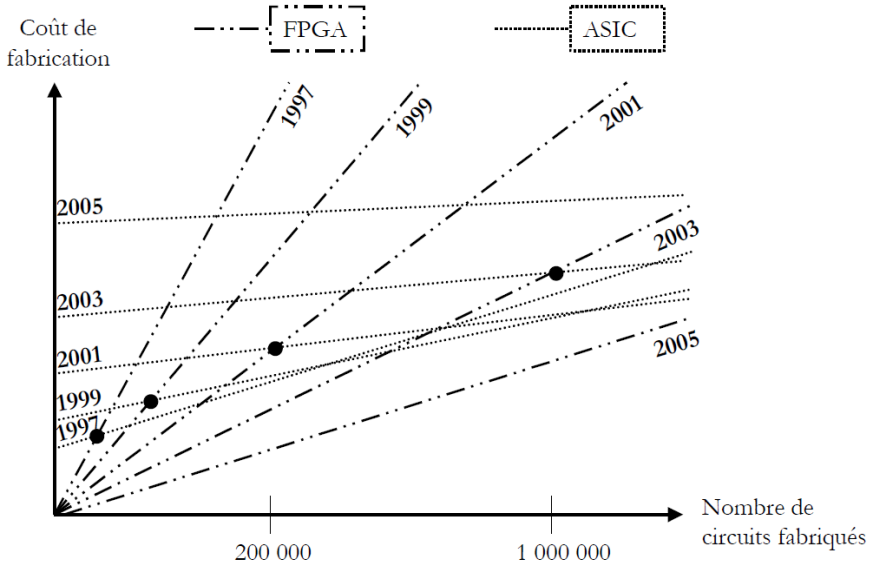


Figure 1 : Evolution des coûts de fabrication des technologies ASIC et FPGA [TrSh03]

Le principal intérêt des FPGA réside dans leur capacité à être reconfiguré, lequel s'accorde parfaitement avec les besoins en termes de flexibilité de la radio logicielle. S'il l'on considère la flexibilité des FPGA, le bond technologique dont ils ont fait l'objet et la baisse de leur coût de fabrication, il n'est pas étonnant de voir leur utilisation se généraliser dans les plateformes de radio logicielle. De nombreuses plateformes radio logicielle embarquent maintenant un ou plusieurs FPGA à l'instar de celle de LYRtech qui embarque deux Virtex 4 [Xili08] ou encore des boitiers USRP N200 et N210 de la société Ettus Research qui embarquent un FPGA Spartan 3A [Ettu11].

Les qualités de la technologie FPGA font de celle-ci une candidate sérieuse pour la fabrication des radios logicielles *sécurisées*. Dans les faits, la conception d'une radio logicielle *sécurisée* est basée sur deux principes fondamentaux. D'une part, la sécurisation des flux d'information traversant la radio. D'autre part, la sécurisation de la radio elle-même. Or,

quel que soit la situation, les mécanismes de protection peuvent être amenés à évoluer dans le temps suite, par exemple, à la mise au point d'une nouvelle attaque matérielle et/ou logicielle. Sur ce point, le FPGA possède un avantage certain par rapport aux ASIC dans la mesure où ils peuvent être reconfigurés pour y implanter (dans une certaine limite) les dernières avancées technologiques en termes de contremesures. Toutefois, cette capacité à être reconfiguré est aussi l'une des principales faiblesses des FPGA. Dans ces conditions, comment s'assurer qu'à chaque instant la configuration d'un FPGA est valide et qu'elle n'a pas fait l'objet de manipulations mal intentionnées ? La question de l'implantation sur FPGA d'une plateforme radio logicielle *sécurisée* ou au moins de ses composantes les plus critiques n'est donc pas une question anodine.

Le *sous-système cryptographique* (ou *Crypto Sub System*) tel que décrit par le supplément sécurité à la SCA [Jtrs04] est l'un de ces composants critiques. Ce dernier est entre autres chargé de la gestion des mécanismes de protection intégrés à une radio logicielle sécurisée ainsi que du chiffrement des flux d'information la traversant. Le niveau de criticité de ce composant en fait un excellent cas d'étude lorsqu'il s'agit de traiter des problématiques d'implantation d'une plateforme radio logicielle *sécurisée* sur FPGA. De façon générale, on pourrait se demander quelle serait l'architecture matérielle idéale de ce composant lorsqu'il est implanté sur FPGA. Comment peut-on tirer profit de la reconfigurabilité des FPGA et de leur structure massivement parallèle ? Pour finir, quelle approche adopter lorsque l'on souhaite garantir le niveau de sécurité de l'implantation d'un tel composant sur FPGA ?

Les travaux présentés dans ce document ont été menés dans le but de répondre aux questions précédentes. Ils suivent une approche descendante en abordant dans un premier temps les questions les plus générales pour ensuite s'intéresser aux plus spécifiques. Ainsi, le premier chapitre présente une architecture de *Crypto Sub System* adaptée à la technologie FPGA. La présentation de cette architecture inclut tout d'abord une analyse des besoins de la radio logicielle sécurisée. Puis, elle apporte des réponses aux problématiques architecturales et à celles de l'isolation fonctionnelle et électrique des circuits au sein d'un FPGA. En outre, l'architecture proposée tire parti des dernières fonctionnalités des FPGA et notamment des techniques de *reconfiguration dynamique partielle*. Les deux derniers chapitres se concentrent sur la conception d'une architecture de composant destiné au chiffrement des flux d'informations traversant la radio logicielle. Une étude théorique de l'implantation sur FPGA d'un cryptoprocesseur multicœur pour la radio logicielle illustrée par une implantation concrète y est présentée. Les aspects matériels aussi bien que logiciels sont abordés dans ces chapitres. Ils apportent aussi quelques éléments de réponses au sujet de l'ordonnancement des tâches cryptographiques sur un *cryptoprocesseur multicœur*.

La présentation de ces contributions est structurée en quatre chapitres de la façon suivante. Tout d'abord, le premier chapitre fournit au lecteur l'ensemble des éléments nécessaires à la compréhension des différentes problématiques relatives à la radio logicielle sécurisée. La première moitié du chapitre présente la notion de radio logicielle et plus particulièrement celle de radio logicielle *sécurisée*. La seconde moitié de ce chapitre aborde la question de l'implantation d'un *Crypto Sub System* sur cible FPGA. Finalement, cette partie conclut sur la

nécessité d'étudier la question de la forme que doit prendre le ou les cryptoprocesseurs intégrés au *Crypto Sub System*.

Pour cela, le second chapitre dresse un état de l'art des travaux publiés portant sur les architectures de cryptoprocesseurs. La première partie de ce chapitre propose un ensemble de critères permettant de caractériser l'adéquation d'une architecture de cryptoprocesseur avec les besoins de la radio logicielle. Suite à cela, un petit nombre de travaux publiés pertinents sont isolés puis étudiés plus en détail dans la seconde partie du chapitre. Pour finir, une synthèse des points forts et faibles de ces architectures est faite afin d'en tirer un ensemble de fils conducteurs qui serviront par la suite de base à la mise au point d'une architecture de cryptoprocesseur adaptée à la radio logicielle sur FPGA.

Les bases d'un cryptoprocesseur adapté à la radio logicielle sécurisée sur FPGA ayant été définies, le troisième chapitre présente de façon théorique la forme que pourrait prendre un tel cryptoprocesseur. A cet effet, les problématiques de *reconfiguration partielle* et *d'isolation électrique* spécifique aux composants FPGA sont initialement abordées. Puis, les principes architecturaux généraux et les problématiques d'interfaçage du *cryptoprocesseur multicœur* sont décrits. Cette description est immédiatement suivie d'une analyse plus approfondie du fonctionnement du cryptoprocesseur et des cœurs cryptographiques qu'il embarque. Pour finir, le chapitre se conclut sur une analyse succincte de la problématique d'ordonnancement des tâches sur ce cryptoprocesseur.

Finalement, la structure théorique d'un cryptoprocesseur dédié à la radio logicielle sur FPGA ayant été présentée, le dernier chapitre s'attache à l'illustrer par une implantation concrète sur FPGA. Dans un premier temps, ce chapitre détaille la façon dont est implanté concrètement sur FPGA le cryptoprocesseur, notamment en présentant les différentes structures matérielles sous-jacentes. Puis, dans un second temps, les résultats d'implantation sur FPGA SRAM Xilinx Virtex 4 et Virtex 6 sont présentés. A ces résultats s'ajoute une étude de performance du cryptoprocesseur en fonction de la charge de travail qui lui est appliquée ainsi que son adéquation avec le chiffrement de flux audio et vidéo. Pour finir, une étude concernant l'utilisation concrète de la reconfiguration partielle dynamique dans le cadre de ce cryptoprocesseur est proposée.

En conclusion, le lecteur trouvera une synthèse des différents travaux présentés dans ce document. Cette synthèse est suivie d'une ouverture traitant de l'ordonnancement des tâches cryptographiques sur les cryptoprocesseurs multicœur et de la difficulté d'implanter sur FPGA les circuits d'interconnexions présents dans les processeurs multicœur.

Chapitre 1

De la radio traditionnelle à la radio logicielle sécurisée

1.1 Introduction

La demande, chaque jour plus importante, sur les marchés des médias et des communications, est à l'origine d'évolutions majeures dans les domaines des technologies de l'information et des communications. Ainsi, la tendance actuelle est à la convergence des moyens de communications. Tendance qui conduit les fabricants à inclure dans leurs produits le support de médias divers et variés et des standards de communication associés. C'est dans ce contexte qu'est apparu le concept de la radio logicielle ; ce dernier ayant fait l'objet de travaux importants à partir des années 90 [CoBo99, Mito95]. La radio logicielle repose sur l'idée simple suivante : une radio dont les caractéristiques sont principalement déterminées par des composants logiciels est à la fois flexible et peu couteuse à construire par rapport à une radio multistandard analogique.

Néanmoins, cette innovation, en rupture avec les méthodes de conception passées, a vu son apparition retardée par un ensemble de verrous technologiques qui n'ont pas encore tous été levés. En effet, l'utilisation de composants logiciels comme organes principaux d'une radio pose de nombreux problèmes notamment en ce qui concerne les performances, la standardisation des matériels et leur sécurité.

Ce premier chapitre se propose de présenter l'ensemble de ces problématiques. Dans un premier temps (sections 1.2 et 1.3), les problématiques liées aux télécommunications ainsi que les solutions apportées par la radio logicielle sont présentées. Puis, dans un second temps, ce chapitre se focalise sur l'aspect sécurité de la radio logicielle en présentant d'abord les menaces visant les réseaux de communications sans fil puis les mécanismes de protection qui y répondent (section 1.4). Par la suite, ce document traite plus particulièrement des mécanismes de protection des données échangées sur le réseau (section 1.5) ce qui débouche sur notre proposition d'architecture sécurisée pour la radio logicielle (section 1.6).

1.2 Contraintes sur les systèmes communicants

Les besoins des utilisateurs de systèmes communicants ont largement évolué ces dernières années. C'est ainsi qu'aujourd'hui les ordiphones sont couramment utilisés pour la navigation sur internet, la lecture de vidéos (streaming), la messagerie instantanée ou, de

façon plus anecdotique, la visio-conférence. Afin que ces services soient réellement utilisables, les infrastructures de communication doivent répondre à certaines contraintes qui sont à l'origine de la notion de *Qualité de Service* (QdS). Nous citerons ici les plus importantes que sont : le débit, les délais de transmission, la gigue et le taux de perte des paquets d'information. A titre d'illustration, le Tableau 1 détaille les contraintes liées aux différents types de flux de données.

Tableau 1: Caractéristiques de flux temps réel en fonction de leur type [SzHa04]

Type de flux	Débit	Latence maximum	Gigue moyenne maximum	Taux de perte maximum
Voix	De 21 à 320 kbps	150 ms	30 ms	1%
Visio conférence	De 128 à 384 kbps	150 ms	30 ms	1%
Streaming vidéo	De 128 kbps à plusieurs dizaines de Mbps	4 à 5 s	NA	5%
Données (internet)	Autant que possible	Quelques secondes	NA	0%

Il s'avère que les contraintes de QdS pèsent particulièrement sur le développement des infrastructures de communication. En effet, ces services nécessitent la mise en place de politiques de gestion de la QdS qui ne sont pas forcément compatibles avec les technologies actuellement disponibles. C'est donc l'usage de ces nouveaux services qui est, en parti, à l'origine de l'évolution incessante des standards de communication.

C'est ainsi que les normes de radiocommunications évoluent rapidement afin d'intégrer un nombre toujours plus grand de services. Par exemple, la première norme de communication mobile pour téléphone portable, Radiocom 2000, a été lancée en 1986 avant d'être abandonnée en Juillet 2000. Par la suite, les technologies GSM[3] (2G), GPRS[4] (2.5G), EDGE[5] et UMTS[6] (3G) sont apparues en l'espace de huit ans. Et cela n'est pas près de s'arrêter, la Corée du Sud vient (fin 2010) de dévoiler un prototype de terminal mobile reposant sur la technologie LTE[7]-avancée (4.5G). Dans les faits, cette technologie vise à fournir des débits 40 fois supérieurs à la 3G (environ 600 Mbps) qui pourront servir à diffuser des contenus 3D en haute définition sur les terminaux mobiles. Bien sûr, la téléphonie mobile n'est pas la seule concernée. La technologie WiFi avec ses multiples versions illustre tout aussi bien l'évolution rapide des standards de communication.

[3] *Global System for Mobile Communication*
[4] *General Packet Radio Service*
[5] *Enhanced Data Rate for GSM Evolution*
[6] *Universal Mobile Telecommunication System*
[7] *Long Term Evolution*

A cela s'ajoute le fait que les standards de communication peuvent aussi varier d'un pays à l'autre. Dans le cas de la norme GSM, cinq sous standards différents coexistent[8] :

- GSM 850 et 1900 : Principalement utilisé aux Etats Unis et au Canada.
- GSM 900 et 1800 : Principalement utilisé en Europe.
- GSM 400 : Destiné au pays faiblement peuplé et en voie de développement pour lesquels les appareils utilisant la norme GSM 900 n'ont pas une portée suffisante.

Ainsi, en plus de devoir changer de mobile à chaque mise à jour des normes, l'utilisateur peut aussi rencontrer des difficultés de communications lorsqu'il change de position géographique. Par conséquent, *l'interopérabilité* et *l'évolutivité* sont devenues des qualités indispensables aux systèmes communicants modernes. En résumé, ceux-ci se doivent d'être *flexibles*.

Les premiers à avoir essayé de répondre concrètement à cette problématique de flexibilité sont les militaires. C'est ainsi qu'au début des années 1990, le programme SPEAKeasy [CoBo99] a vu le jour. Il s'agissait alors de développer une radio compatible avec l'ensemble des standards de communications utilisés par les différents corps de l'armée américaine, chacun d'eux utilisant des bandes de fréquence et des modulations différentes. Le programme SPEAKeasy reposait sur l'utilisation du principe de la radio logicielle qui consiste à baser la structure d'une radio sur architecture matérielle générique et programmable en fonction des besoins. Néanmoins, si la contrainte de flexibilité est bien à l'origine de la philosophie de l'architecture de SPEAKeasy, il en existe une autre que l'on ne peut négliger lorsque l'on parle d'applications militaires, c'est la *sécurité*.

C'est ainsi que les travaux de recherche sur la sécurisation de l'architecture de SPEAKeasy ont été les précurseurs d'un grand nombre d'études portant sur la sécurisation des systèmes de communication basés sur le principe de la radio logicielle [BoMK08, KuBF05, MaNM08, MuMa06]. De façon générale, la sécurisation des moyens de communication pose de nombreux problèmes de conception. Premièrement, étant donné la diversité des types d'attaques existants (attaques matérielles, logicielles, ingénierie sociale, …), la mise en place de mécanismes de sécurité au sein des systèmes de communication est une problématique complexe qui nécessite, le plus souvent, des solutions avancées. A titre d'illustration, le lecteur pourra se reporter au standard de radio logicielle sécurisée proposé par le gouvernement américain [Jtrs04]. A cela s'ajoute le fait que les standards cryptographiques militaires varient d'un pays à l'autre. Deuxièmement, ces mécanismes de protection ont, la plupart du temps, un impact sur les performances des systèmes qu'ils protègent. Par exemple, un système de chiffrement ajoutera inévitablement un délai, aussi infime soit-il, au traitement d'un flux d'information [RaTa10]. Or, nous avons vu que la qualité de service de la VoIP ou de la visio-conférence dépend des délais de transmission

[8] C'est cette multitude de standards qui a été à l'origine de l'appellation bi-bande ou tri-bande de certains téléphones.

de l'information. Pour finir, *sécurité* et *facilité d'utilisation* sont souvent des notions antagonistes. Ainsi, si l'utilisation de mots de passe est nécessaire lorsqu'il s'agit de sécuriser certains services informatiques, cela n'a pas pour effet d'améliorer leur confort d'utilisation, bien au contraire. En définitive, un bon système de sécurité se doit d'être aussi transparent et simple que possible tout en répondant aux contraintes du cahier des charges. Par la suite, nous reviendrons plus longuement sur cette contrainte qu'est la sécurité.

En résumé, les concepteurs d'un système de communication moderne doivent prendre en compte les contraintes suivantes :

- **Performances** (débit, latence, taux de perte des paquets de données, …)
- **Flexibilité** (évolutivité, interopérabilité, …)
- **Sécurité** (confidentialité, intégrité et authentification)
- **Fiabilité** (principalement pour les systèmes militaires et/ou critiques)
- **Consommation** et **compacité** (lorsque le système est mobile)
- **Coût**

Bien que la question de flexibilité soit centrale lors de la conception d'une radio logicielle, la question de la sécurité ne peut être négligée pour les applications militaires. C'est donc cette contrainte de sécurité qui est à l'origine de nombreux choix techniques et architecturaux effectués lors du développement de la radio SPEAKeasy. Etant financé par la DGA, les travaux présentés dans ce manuscrit portent à la fois sur la sécurité des radios logicielles, leur flexibilité et les performances des crypto-systèmes qui les composent.

Avant de continuer plus en avant l'étude de la radio logicielle sécurisée, la section suivante s'attache, dans un premier temps, à présenter les principes de la radio traditionnelle et ceux de la radio logicielle. Puis, dans un second temps, à montrer en quoi la radio logicielle permet de répondre, au moins en partie, aux contraintes formulées précédemment.

1.3 Radio traditionnelle et radio logicielle : principes, avantages et défauts

1.3.1 Principe de la radio traditionnelle

Jusqu'au début des années 90, la logique de conception des architectures radios ne mettait pas l'accent sur la flexibilité des architectures développées. Les couches matérielles et logicielles des radios traditionnelles n'étaient ni conçues pour être génériques afin d'être facilement réutilisables, ni conçues pour être évolutives au cours du temps. En général, la radio était composée d'une juxtaposition de chaines de modulation/démodulation, une pour chaque standard. Globalement, une radio traditionnelle peut être divisée en trois couches illustrées sur la Figure 2, qui sont :

- *Le front-end analogique* : Il permet la conversion du signal radio fréquence (RF) capturé par l'antenne en un signal analogique dont le spectre de fréquence est

centré sur une fréquence intermédiaire (FI) suffisamment faible pour que le signal puisse être échantillonné par un convertisseur analogique/numérique.

- *La couche numérique* : Cette couche en aval du convertisseur analogique/numérique permet un traitement numérique de l'information. Elle peut être composée de circuits programmables (ex : DSP) ou non. Toutefois, les programmes s'exécutant sur ces composants ne sont pas prévus pour être portables et leur conception ne suit aucun standard particulier.
- *La couche logicielle haut niveau* : Celle-ci fournit les services nécessaires à l'interface homme-machine. Elle n'a que peu d'effet sur le fonctionnement interne de la radio.

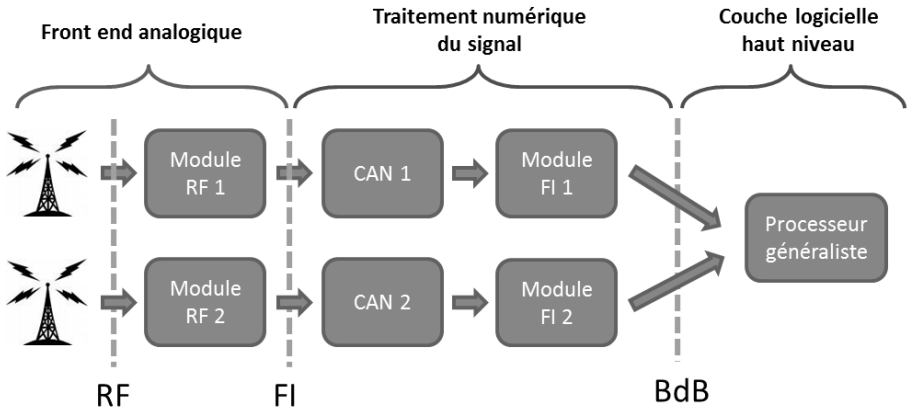


Figure 2 : Récepteur radio multistandard classique

A l'intérieur de ce type de récepteur, il existe plusieurs chaines de réception/émission, chacune d'entre elles est spécifique à la norme ou à la bande de fréquence traitée et chaque chaine est connectée au processeur central. Ce dernier a pour tâche principale la configuration des chaines de communication ainsi que leur interfaçage avec les périphériques d'E/S (ex : périphériques audio et vidéo, interfaces réseaux ou panneaux de contrôle). Cette architecture à l'avantage de pouvoir fonctionner à des fréquences élevées car le matériel est optimisé pour l'application ciblée. Malheureusement, elle a comme défaut majeur son manque de flexibilité. Effectivement, dans le cas d'une architecture radio traditionnelle, il faut prévoir autant de chaines de modulation/démodulation en bande de base que de normes de communication utilisées. De plus, il est par la suite impossible d'enlever ou d'ajouter à la radio la gestion d'une norme particulière. Cela a pour principales conséquences d'augmenter les coûts de fabrication, la consommation en énergie et l'encombrement des systèmes de communication.

1.3.2 **Principe de la radio logicielle**

Afin d'apporter des réponses à ces problématiques, le concept de radio logicielle (ou *Software Defined Radio*) a vu le jour. En résumé, le principe de la radio logicielle est de réaliser de façon logicielle le plus grand nombre possible de fonctionnalités et notamment la modulation et la démodulation des signaux radios.

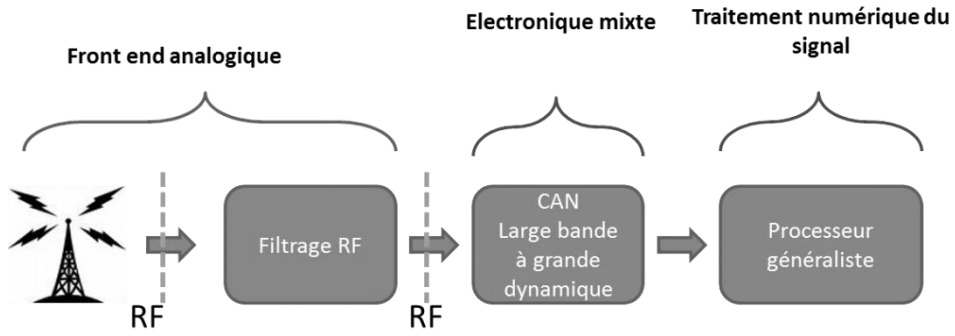


Figure 3 : Récepteur radio logiciel idéal

La Figure 3 détaille l'architecture du récepteur d'une radio logicielle idéal dans lequel l'ensemble du traitement du signal radio à l'exception du filtrage RF est effectué de façon logicielle. Au sein d'une telle architecture, la sortie RF de l'antenne est directement échantillonnée et numérisée, après filtrage RF, par un *convertisseur analogique numérique* (CAN) large bande ayant une dynamique élevée puis, la sortie du CAN est traitée par un *processeur généraliste* (GPP). Les modules RF et FI d'une radio classique sont décrits sous forme logicielle par les programmes s'exécutant sur ce processeur. Toutefois, ce type d'architecture pose plusieurs problèmes. La principale limitation vient des technologies de conversion analogique/numérique qui concilient difficilement large bande et résolution élevée [RDBD08]. En effet, les protocoles de communication sans fil utilisent des fréquences porteuses qui peuvent aller de quelques mégahertz à plusieurs gigahertz pour les communications satellites. Par ailleurs, en admettant qu'il soit possible de réaliser un tel CAN, il faudrait aussi que le GPP qui y est connecté puisse traiter les données de façon suffisamment rapide. Or, les GPP les plus performants fonctionnent à une fréquence de l'ordre du gigahertz qui n'est pas suffisante pour traiter des signaux radios RF dans la mesure où ils n'ont pas une architecture adaptée à cette tâche. Dans ces conditions, une radio logicielle idéale n'est pas réalisable et plusieurs *front-end* analogiques doivent être encore utilisés pour traiter les signaux hautes fréquences issus des antennes.

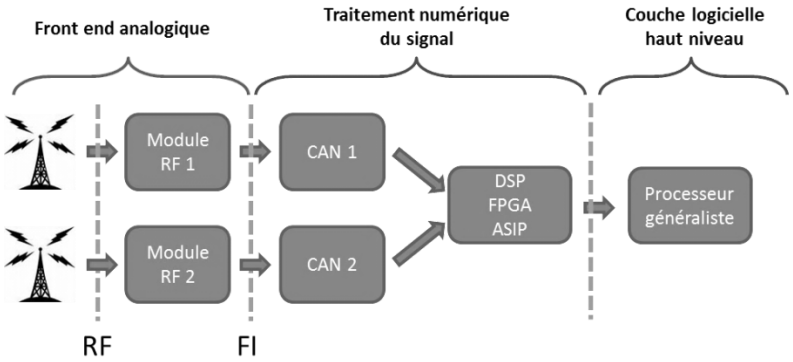


Figure 4 : Récepteur radio logiciel restreint

Concrètement, une solution intermédiaire (Figure 4) mêlant à la fois *front-end* analogiques, composants numériques *programmables* ou *configurables* aussi bien spécialisés que généralistes est utilisée. Une telle architecture transpose d'abord les signaux RF en fréquence intermédiaire puis les échantillonne et les traite en utilisant des composants numériques basés sur des architectures massivement parallèles. En bout de chaine, on retrouve un processeur généraliste pour effectuer l'interfaçage entre les chaines de communication et les périphériques d'interface homme-machine ou d'entrée/sortie de la radio.

La partie logicielle d'une radio logicielle peut être décomposée en plusieurs couches. La Figure 5 représente cette décomposition. Les architectures radio logicielle évoluées reposent sur ce type de décomposition.

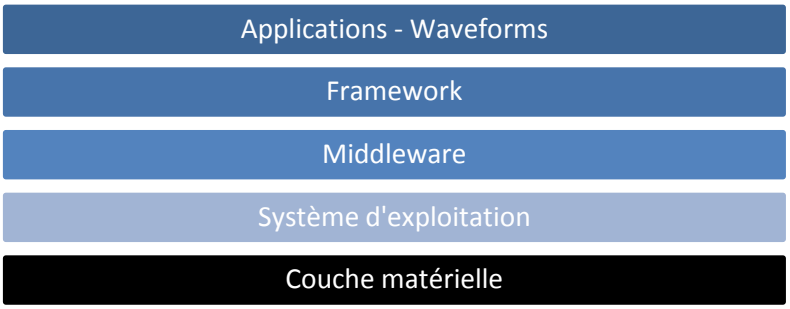


Figure 5 : Décomposition en couches d'une radio logicielle

L'objectif d'une telle décomposition est de permettre la portabilité et la réutilisabilité des composants logiciels en isolant les couches logicielles de haut niveau du matériel sur lequel elles s'exécutent. L'abstraction du matériel se fait via l'OS et le middleware. Le middleware (ex. CORBA[9]) permet de distribuer de façon transparente l'exécution des applications sur plusieurs unités de calcul. Quant au *Framework,* celui-ci permet de fournir au développeur un support pour l'implantation des composants logiciels qui constitueront la radio logicielle. La *Software Communication Architecture* (SCA) [Jtrs06] est l'un des frameworks les plus connus dans le domaine de la radio logicielle. Développée par le gouvernement américain au début des années 2000, cette architecture spécifie les principes de fonctionnement d'une radio logicielle, c'est-à-dire : la manière par laquelle les logiciels constituant *les formes d'ondes*[10] sont chargés, la manière de les exécuter et la manière de les faire communiquer entre eux. L'effet induit par la standardisation de ces mécanismes est une plus grande facilité lors du portage d'une forme d'onde depuis une radio logicielle vers une autre.

Dans ces premières versions, la SCA reposait largement sur l'utilisation du middleware CORBA qui permet de distribuer un ensemble de tâches logicielles sur plusieurs unités de

[9] *Common Object Request Broker Architecture*

[10] Une *forme d'onde* correspond à un processus complet de traitement d'un signal : capture, émission, réception puis restitution.

calcul. Par la suite, la dernière version de la SCA [Jtrs10] a pris ses distances avec CORBA qui est parfois considéré comme trop couteux en termes de ressources pour les applications embarquées ou pas assez performant pour les applications temps réel [SACA08]. Il existe actuellement plusieurs implantations de la SCA et notamment SCARI [JXJB09] et OSSIE [GDSV09]. De plus amples informations sur la SCA sont disponibles dans la dernière version du standard [Jtrs10].

1.3.3 **Avantages et défauts de la radio logicielle sur la radio traditionnelle**

Si l'on reprend les contraintes définies dans la partie 1.2 et que l'on compare les qualités et défauts des radios traditionnelles et des radios logicielles, on constate qu'en ce qui concerne les performances en termes de débit, de latence et de consommation d'énergie, la radio traditionnelle possède naturellement une longueur d'avance sur la radio logicielle. En effet, une radio classique multistandard embarque une chaine de codage/décodage spécifiquement conçue pour chaque norme de communication et cela a pour effet de maximiser les performances. A l'inverse, le concept de la radio logicielle tend à mutualiser autant que possible les ressources matérielles en implantant chaque norme au niveau logiciel. Les ressources matérielles étant génériques, les performances sont moindres. Toutefois, l'utilisation de circuits numériques programmables tels que les FPGA (*Field Programmable Gate Array*) a permis une amélioration substantielle des performances des radios logicielles tout en préservant leur flexibilité [CoBo99].

Par contre, la radio logicielle se démarque lorsqu'il s'agit des contraintes de flexibilité, compacité et coût. En effet, de par son caractère programmable, la radio logicielle est hautement flexible. De plus, la mutualisation des ressources matérielles entre les différents standards de communications implantés permet d'une part, de réduire le nombre de composants embarqués dans une radio et d'autre part, de réduire le coût de fabrication d'une radio logicielle dont la partie matérielle est générique.

Pour finir, en ce qui concerne la sécurité des radios logicielles, deux aspects s'opposent. D'un côté, le caractère programmable d'une radio logicielle fait que celle-ci est potentiellement plus sensible aux attaques qu'une radio classique. A contrario, il est possible de mettre à jour une radio logicielle afin de combler une faille de sécurité alors que ce n'est pas forcément le cas avec une radio classique dont la structure est en grande partie figée. Cet aspect de la radio logicielle a fait l'objet d'un grand nombre de publications traitant de la mise au point de techniques de mise à jour sécurisée [ShUA00, SuMy05, UcUK00].

En résumé, le caractère flexible de la radio logicielle apporte une réponse aux problématiques d'interopérabilité et d'évolutivité. Quant aux défauts de la radio logicielle, la plupart ont pour origine des verrous technologiques qui tendent peu à peu à disparaitre. En réalité, lorsqu'on considère l'aspect sécurité des technologies de l'information, la principale faiblesse de la radio logicielle réside en ce qui fait sa force : son caractère programmable. Une radio logicielle n'est donc pas à l'abri d'attaques logicielles utilisant, par exemple, l'installation de mises à jour corrompues. La question de la sécurité des radios logicielles,

qui est traitée dans la partie suivante, fait donc l'objet d'une attention toute particulière de la part des militaires.

1.4 Origines et principes de la radio logicielle sécurisée

1.4.1 Classification des menaces

Qu'ils soient militaires ou civils, les réseaux de communication doivent faire face à un certain nombre de menaces. Bien sûr, la mise en œuvre d'attaques varie en fonction des circonstances et du type de réseau ciblé. Toutefois, on peut les classifier de la manière suivante :

- Attaques ciblant la couche protocolaire :
  - ***Ecoute :*** Ce type d'attaque consiste à intercepter une communication entre plusieurs interlocuteurs afin d'en tirer profit. Bien que passive par nature, une attaque reposant sur l'*écoute* peut nécessiter l'exécution d'une attaque active (ex : récupérer une clé de chiffrement). Les réseaux sans fil sont particulièrement vulnérables face à ce genre d'attaque, car les signaux radios se propagent librement dans l'air. A titre d'illustration, on pourra citer le standard WEP (*Wired Equivalent Privacy*) qui est un cas d'école lorsque l'on parle des protocoles sensibles aux attaques par *écoute* [FlMS01, StIR04, TeBe09].
  - ***Attaque par rejeu :*** Ce type d'attaque consiste à réémettre un message précédemment envoyé sur le réseau. Par exemple, si l'on admet qu'un paquet d'authentification contient un mot de passe chiffré permettant l'accès à un service donné, l'attaquant peut intercepter le paquet puis le réémettre plus tard afin d'accéder à ce service. De la même façon, on peut imaginer que lors de la mise à jour logicielle d'un appareil, un attaquant rejoue une ancienne mise à jour contenant une faille de sécurité [UcUK00].
  - ***Attaque « man in the middle » :*** Une telle attaque a lieu lorsque l'attaquant se place entre deux parties légitimes d'un réseau et qu'il usurpe l'identité de chacune des parties sans que l'autre ne s'en rende compte. Cela lui permet d'injecter ou de supprimer des messages lors d'un échange d'information ou bien de mener une attaque par écoute en restant passif. Contrairement aux réseaux filaires, les réseaux sans fil sont particulièrement sensibles à ce type d'attaque étant donné qu'il est virtuellement possible de se connecter à n'importe quel réseau sans fil du moment que l'on est a porté des autres utilisateurs [MeWe04].
  - ***Usurpation d'identité :*** L'usurpation d'identité consiste pour un attaquant à s'attribuer l'identité ou les droits d'un utilisateur légitime afin de tromper les autres utilisateurs du réseau [BaHK06]. L'usurpation d'identité est principalement utilisée comme première étape d'une attaque *man in the middle*.
  - ***Déni de service :*** Les attaques par déni de service visent à interrompre la fourniture d'un service réseau quelconque. Elles peuvent être très simples ou très complexes. En ce qui concerne les réseaux sans fil, il est possible de

brouiller les échanges radios sur une certaine bande de fréquence. Toutefois, il est facile de prévenir ce type d'attaque, car l'attaquant doit être proche de sa cible. Il existe de nombreuses autres techniques d'attaques plus difficiles à contrer. Parmi celles-ci, on peut citer : les attaques par bombardement de paquets, les attaques profitant d'une faiblesse d'un protocole réseau [BeSa03] ou les attaques par usurpation d'identité afin de dévier le trafic à destination d'un utilisateur légitime.

- Attaques ciblant la couche logicielle :
  - ***Utilisation de failles d'implantation :*** Ce type d'attaque consiste à utiliser des failles dans l'implantation d'un système afin d'y obtenir un accès illégitime. Un utilisateur peut par exemple se servir d'une faille présente dans le noyau du système d'exploitation pour d'obtenir les droits d'un super utilisateur. Les API (*Application Programming Interface*) cryptographiques font aussi l'objet de ce type d'attaque lorsqu'elles sont mal définies et/ou mal implémentées [BoAn01].
  - ***Logiciels malicieux :*** Ce type d'attaque consiste à introduire un logiciel malicieux dans un système cible soit en utilisant une faille d'implantation soit en usant de la crédulité de l'utilisateur du système. En ce qui concerne la radio logicielle, le mécanisme de mise à jour est le principal vecteur d'infection par des logiciels malicieux. Une fois le logiciel introduit, celui-ci peut récupérer des informations confidentielles et les envoyer à l'attaquant ou, par exemple, affilier le système à un Botnet[11].
- Attaques ciblant la couche matérielle :
  - ***Attaque par canaux cachés :*** Ces types d'attaques consistent à étudier de manière non intrusive le comportement d'un système matériel afin d'en extraire un secret. En effet, la consommation en énergie d'un composant, le rayonnement électromagnétique qui en émane ou les temps d'exécution des programmes peuvent être corrélés avec des informations confidentielles. Il s'ensuit qu'un système matériel non-protégé peut être à l'origine de fuites d'informations involontaires. Par exemple, le temps d'exécution de l'algorithme d'exponentiation rapide utilisé par RSA est une fonction affine du nombre de bit à 1 dans la clé secrète. Ce type d'attaque a fait l'objet d'un grand nombre de travaux, le lecteur pourra se reporter à [GMND11, Goub01, Stan10] pour avoir un aperçu des différentes techniques existantes.
  - ***Attaque par « probing » :*** Ce type d'attaque consiste à insérer une sonde dans le matériel attaqué afin d'en extraire une information confidentielle. On peut, par exemple, utiliser un keylogger pour capturer les entrées clavier. De façon plus invasive, un attaquant peut aussi capturer les signaux entre un processeur et les périphériques qui y sont reliés et notamment sa mémoire vive.
  - ***Attaque par injection de fautes :*** Les attaques par injection de fautes sont des attaques nécessitant d'avoir accès au matériel cible. Elles peuvent être

[11] Réseau de machines constitué le plus souvent dans un but malveillant.

invasives si le composant cible est en partie détruit par l'attaque ou non invasives si aucune modification n'y est apportée. Elles consistent à injecter des données invalides dans un système afin de le détourner de son fonctionnement normal et d'en extraire des informations confidentielles. Les fautes peuvent être injectées en utilisant les broches du composant cible. On peut, par exemple, envoyer des données mal formatées en adaptant les techniques de *fuzzing* [SuGA07] couramment utilisées dans le domaine de la sécurité des logiciels. On peut aussi faire varier la tension d'alimentation d'un composant, sa fréquence d'horloge ou sa température durant son fonctionnement [BCNT06]. La technologie laser est aussi utilisée pour injecter des fautes dans un circuit [Skor09, TrKo10]. Les techniques d'injection de fautes peuvent nécessiter la mise à nu du circuit électronique en enlevant le package qui l'entoure [Wein00]. Cette dernière méthode fortement intrusive conduit à la destruction partielle, voire totale, du composant.

1.4.2 **Mécanismes de protection**

Une fois les types de menaces identifiés et classifiés, il est possible d'entamer une réflexion sur les moyens qui peuvent être mis en œuvre pour protéger un système contre tel ou tel type de menaces. De façon générale et quel que soit le type de réseau considéré, la protection passe par l'utilisation de protocoles de communication robustes et sécurisés, l'exécution de logiciels sûrs et fiables sur les nœuds du réseau et la protection physique des nœuds du réseau.

En ce qui concerne l'aspect protocolaire, les mesures de protection passent par l'utilisation de services cryptographiques tels que le chiffrement, l'authentification et la vérification de l'intégrité des données échangées. Le chiffrement permet de garantir la confidentialité des données échangées lors d'une communication. En général, les messages sont chiffrés grâce à des algorithmes de chiffrement symétriques. L'authentification qui inclut aussi l'intégrité permet de s'assurer de la provenance d'un message. Elle repose sur l'utilisation d'un algorithme de chiffrement symétrique dans un mode qui permet l'authentification des messages ou sur le chiffrement d'un condensé (généré par une fonction de hachage) du message par un algorithme de chiffrement asymétrique. A titre d'exemple, les standards WiFi et IPsec utilisent des algorithmes cryptographiques tels qu'AES-CCM [Nist04] et AES-GCM [Nist07] qui permettent à la fois de chiffrer et authentifier des paquets de données. Au contraire, l'utilisation de méthodes d'authentification basées sur un algorithme de hachage (ex : SHA) et un algorithme asymétrique du type RSA ou ECC est plutôt réservée à la signature de documents. En effet, elles ne nécessitent pas le partage d'une clé secrète entre expéditeur et destinataire. Par ailleurs, les algorithmes de chiffrement asymétriques sont moins rapides que les algorithmes symétriques, ils ne sont donc pas utilisés pour le chiffrement de paquet en temps réel. Toutefois, les services cryptographiques n'apportent qu'une réponse partielle aux problématiques de sécurité, car pour qu'un protocole de communication soit sûr, il faut aussi qu'il soit robuste. Par exemple, le standard IP est par nature peu robuste, car il fait l'objet de problèmes de

conception permettant la mise en œuvre d'attaques par déni de service [MiRe04]. Un protocole robuste doit limiter la possibilité de mener des attaques par déni de service ou des attaques du type « man in the middle » [SiSe10]. Outre les services cryptographiques et autre protocole robuste, on peut aussi noter l'utilisation de techniques d'obfuscation des signaux radios. Par exemple, l'étalement de spectre [PiMS91] est une technique d'obfuscation couramment utilisée par l'armée.

En ce qui concerne le matériel radio en lui-même, il s'agit d'une part de garantir le bon fonctionnement des logiciels qui s'exécutent dessus et d'autre part, de le protéger contre d'éventuelles attaques matérielles. La protection des logiciels s'exécutant sur la radio est assurée par des moyens tels que l'utilisation de systèmes d'exploitation sécurisés [MuMa06] ou d'outils de détection des codes malveillants. Côté matériel, les efforts de développement portent particulièrement sur la protection des systèmes de communication contre les attaques par canaux cachés [JSMY09] et les attaques par injections de fautes. A cet effet, des techniques limitant les fuites d'informations involontaires [BaMV05] ou d'autres visant à durcir les circuits électroniques ont été développées [LAMT07].

Par la suite, ces travaux se limiteront à l'étude d'architectures dédiées à la fourniture des services cryptographiques des couches protocolaires et ce pour les raisons suivantes : En ce qui concerne les problématiques de durcissement de circuit ou de sécurité logicielle, celles-ci sortent manifestement du cadre général de cette thèse qui est l'électronique numérique. Ensuite, il s'avère que les contremesures destinées à la protection contre les attaques par canaux cachés sont le plus souvent spécifiques à un algorithme ou une technologie particulière. Or, cette spécificité n'est pas compatible avec l'approche haut-niveau adoptée par ce travail. En outre toutes ces problématiques font déjà l'objet de recherches intensives.

1.4.3 **Le supplément sécurité à la Software Communication Architecture**

Bien qu'initialement développée par l'armée américaine, la SCA n'intègre aucun mécanisme permettant de protéger la radio sur laquelle elle est implémentée. En effet, dans un premier temps, les efforts des développeurs se sont principalement concentrés sur la mise au point d'architectures radio logicielle fonctionnelles. Toutefois, lorsqu'il a été question de sécuriser les architectures basées sur le concept de la radio logicielle, il a fallu développer un cadre de travail formel prenant en compte l'ensemble des mécanismes de protection précédemment présentés ainsi que les spécificités propres à la radio logicielle. C'est dans ce but que le gouvernement américain initia une réflexion sur le sujet qui aboutit en 2004 à la publication d'un supplément sécurité à la version 2.2.1 de la SCA [Jtrs04]. A l'heure actuelle, ce supplément sécurité n'est plus supporté et aucune mise à jour ne semble prévue. Néanmoins, de nombreux travaux de recherches [MaNM08, MuMa06, Turn05] reposent sur ces spécifications. En parallèle des travaux américains, les européens, via le programme ESSOR[12] (*European Secure Software defined Radio*), ont travaillé sur la mise au point de leur

[12] *http://www.eda.europa.eu/*

propre supplément sécurité à la SCA. Hormis les grandes lignes du programme, très peu d'informations ont jusqu'à présent filtré.

Une radio conforme à la SCA sécurisée doit fournir un certain nombre de services qui permettent d'assurer sa sécurité. Ces services doivent être pour la plupart implémentés de façon hybride, c'est-à-dire qu'ils doivent reposer sur des mécanismes logiciels mais aussi matériels. De cette façon, leur résistance aux attaques logicielles est accrue.

Parmi ces services, on trouve principalement :

- Les services cryptographiques standards (chiffrement, intégrité et authentification).
- La mise en œuvre de politique de sécurité.
- La gestion des clés et des certificats cryptographiques.
- Le contrôle des accès à la radio.
- La gestion de la configuration du matériel.
- La mise en œuvre d'alarmes et de systèmes d'audits.
- La gestion et la sécurisation de la mémoire.

Afin de garantir une mise en œuvre à la fois sûre et standardisée de ces services, le *Supplément Sécurité à la SCA* est basé sur deux concepts centraux : le concept *d'architecture en coupure* et un ensemble d'interfaces et de spécifications permettant de standardiser les mécanismes mis en œuvre pour sécuriser une radio reposant sur la SCA. Il s'ensuit que le supplément sécurité fournit une méthodologie de conception plutôt que des détails d'implantations qui eux restent à la charge du développeur.

- *Interfaces du supplément sécurité à la SCA :* Ces interfaces permettent la gestion des composants garantissant la sécurité d'une radio compatible avec la SCA sécurisée. Elles peuvent être implantée en utilisant un middleware tel que CORBA lorsque la cible matérielle est un composant compatible (ex : un processeur généraliste). Lorsque ce n'est pas le cas, des mécanismes de communication spécifiques doivent être mis en place afin de permettre le contrôle des composants qui ne sont pas compatibles CORBA.
- *Architecture en coupure :* Un circuit basé sur le principe d'architecture en coupure est divisé en deux parties : une zone *rouge* et une zone *noire*. La zone *rouge* traite l'ensemble des données confidentielles qui ne sont pas chiffrées (ex. voix en sortie d'un micro). La zone *noire* traite les informations non-confidentielles ou chiffrées qui sont reçues et transmises par la radio. L'architecture en coupure rouge/noire implique une isolation fonctionnelle et électrique entre les zones rouge et noire. L'isolation électrique entre les zones rouge et noire permet de limiter la probabilité qu'une information passe de la zone rouge à la zone noire via un canal caché (consommation de puissance, rayonnement électromagnétique). L'isolation fonctionnelle (cryptographique) est fournie par un composant qui joue le rôle d'une

interface sécurisée entre les zones rouges et noires. Ce dernier est nommé *Crypto Sub System* (CSS) dans le cadre du supplément sécurité à la SCA.

Les parties suivantes détaillent plus amplement le fonctionnement du CSS avant d'en proposer une implantation.

1.5 Le Crypto Sub System (CSS)

1.5.1 Le CSS vu comme une passerelle sécurisée

Le supplément sécurité à la SCA décrit le CSS comme une passerelle sécurisée par laquelle passe toutes les données échangées entre les zones rouge et noire. Les données échangées transitent au sein du CSS en empruntant un ou plusieurs canaux cryptographiques chacun d'eux possédant ses propres clés et protocoles cryptographiques. Chaque canal est identifié par un identifiant unique. Les données échangées sont encapsulées dans des messages CORBA, composés d'un en-tête et d'un corps. Lorsqu'un message CORBA traverse le CSS, seul le corps du message a besoin d'être chiffré. Son en-tête, quant à elle, est envoyée en texte clair afin que le *module de bypass* du CSS puisse en vérifier la validité. Ainsi donc, le CSS (Figure 6) est divisé en deux parties : un *module de bypass* et un *module cryptographique*.

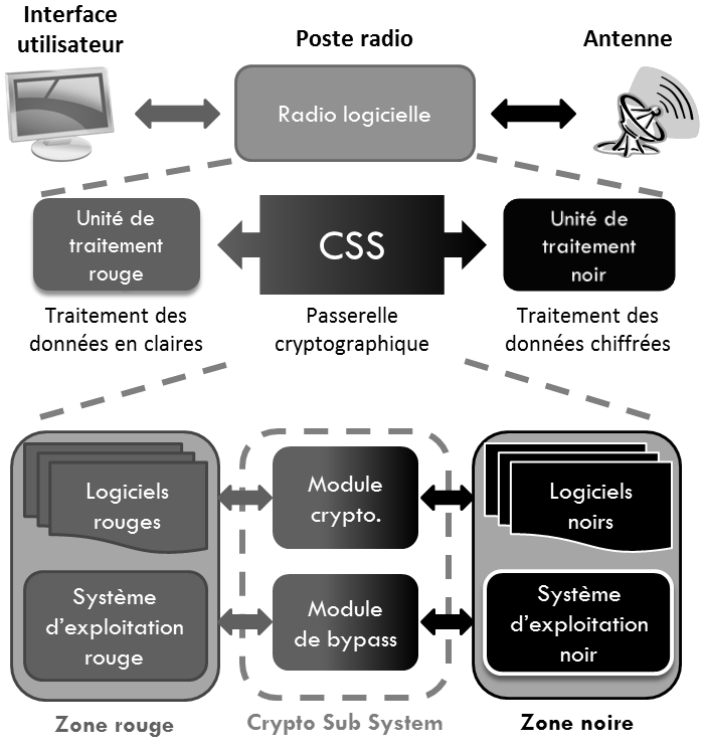


Figure 6: Le CSS en tant que passerelle

1.5.2 **Caractérisation des messages traversant le CSS**

Le CSS doit traiter trois grandes familles de messages. En voici les caractéristiques :

- ***Messages propres à CORBA (Type 1) :*** Ce type de message est utilisé par les composants du framework CORBA afin qu'ils puissent communiquer entre eux. Par exemple, le composant *NamingService* de CORBA liste l'ensemble des composants CORBA disponibles au sein d'une architecture CORBA donnée. Lorsqu'un client tente d'établir une connexion vers un composant particulier, il doit en premier lieu envoyer une requête de *Type 1* au *NamingService* CORBA afin d'obtenir une référence pointant vers le composant recherché. En outre, le *NamingService* peut être situé en zone rouge tandis que le client est situé en zone noire. Par conséquent, le CSS doit pouvoir traiter des messages de Type 1.
    - ***Messages propres à la SCA (Type 2) :*** Ces messages permettent aux composants logiciels de la SCA de communiquer entre eux. Ce type de message est utilisé, par exemple, lorsqu'un composant SCA de la zone rouge a besoin de communiquer avec un composant SCA de la zone noire lors de l'initialisation de la radio logicielle.
    - ***Messages propres à une forme d'onde :*** Ces messages correspondent à des flux de données transitant au sein de la radio logicielle (ex : voix, vidéo, …).

Le supplément sécurité ne précise pas exactement les mécanismes utilisés pour transmettre les messages de Type 1 et 2. Toutefois, les interfaces proposent plusieurs fonctions (par exemple *Encrypt*, *Decrypt* ou *Transform*) qui standardisent la transmission des messages de Type 3 au sein du CSS. Quoiqu'il en soit, étant donné que chacun de ces types de message transportent des données différentes, ils doivent faire l'objet de traitements distincts (cf. Tableau 2).

Tableau 2: Modules utilisés en fonction du type de message

Type de message	Module de Bypass	Module Cryptographique
Type 1	Oui	Non
Type 2	Oui	Non
Type 3	Oui	Oui

1.5.3 **Architecture fonctionnelle du CSS**

Le supplément sécurité suggère le principe d'une architecture pour le CSS. L'architecture proposée repose sur trois modules principaux : un module de bypass, un ou plusieurs modules cryptographiques et un module de gestion appelé *Security Manager*. La Figure 7 illustre l'architecture du CSS proposé par le supplément sécurité à la SCA [Jtrs04]. La suite de cette partie présente succinctement chacun de ces modules.

- ***Le module de Bypass :*** Toutes les données qui n'ont pas à être chiffrées ou déchiffrées doivent transiter par le *module de Bypass* lorsqu'elles traversent le CSS. Ce module filtre des données telles que des données de contrôle (ex : en-tête GIOP) ou des données utilisateurs de Type 3. Les données sont filtrées selon un ensemble

de règles formant une politique de sécurité. Afin de vérifier la validité d'une donnée, le module vérifie : le type de la donnée, sa source et sa destination. Le module de Bypass rejette toute donnée pour laquelle il n'existe pas une règle de sécurité valide. Pour cela, il rejette le message, ferme le canal de transport et prévient l'utilisateur.

- ***L'unité cryptographique :*** L'unité cryptographique permet de chiffrer et déchiffrer les corps de messages, d'authentifier les messages entrant et de vérifier leur intégrité. Elle permet aussi de déchiffrer et authentifier les composants logiciels qui seront chargés par la radio. Il est à noter que l'unité cryptographique ne peut accéder aux clés et certificats cryptographiques qu'en lecture seule.
- ***Le Security Manager :*** Ce composant assure la configuration, la gestion et la bonne marche (en termes de sécurité et fiabilité) du CSS. Le *Security Manager* gère, par exemple, les clés et certificats cryptographiques, les politiques de sécurités et le fonctionnement des alarmes. Les clés et certificats cryptographiques sont chargés sous forme chiffrée par le *Security Manager* au travers d'un port sécurisé (*Fill I/O*). Ils sont ensuite stockés dans une mémoire interne au CSS. Le *Security Manager* peut être contrôlé via les interfaces du supplément sécurité à la SCA.

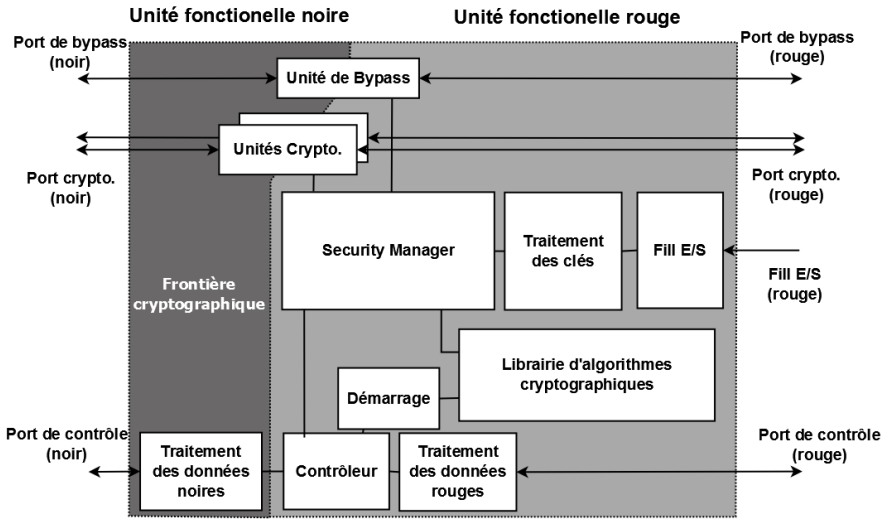


Figure 7: Architecture fonctionnelle du CSS [Jtrs04].

1.6 Proposition d'une architecture reconfigurable de CSS

1.6.1 Motivation de ce travail

Il existe, aujourd'hui, un petit nombre de composants électroniques qui ont été spécifiquement conçus pour jouer (au moins en partie) le rôle de CSS dans les équipements de communication. Ces composants sont fabriqués par des entreprises privées telles que Harris Corp [Harr05], General Dynamics [Gene06] ou Raytheon [Rayt00] et prennent la forme de SoC basés sur des technologies ASIC. Cela pose plusieurs problèmes : d'une part

les circuits basés sur des technologies ASIC manquent de flexibilité et d'autre part, ces constructeurs ne publient pas leur architecture pour des raisons de propriété intellectuelle et de sécurité militaire. Toutefois, il semble évident qu'à l'heure de l'internet, la sécurité des communications n'est pas une problématique propre au domaine militaire. En outre, le principe de Kerckhoffs [Kerc83] montre que la sécurité par le secret n'est pas LA solution bien qu'elle puisse s'avérer efficace dans un certain nombre de cas. Ces deux constations sont à l'origine de la proposition d'une architecture ouverte de CSS.

Etant donné la contrainte de flexibilité à laquelle fait face la conception d'une radio logicielle, nous avons choisi de cibler les plateformes FPGA SRAM pour nos implantations. En effet, la flexibilité est une conséquence directe de la capacité qu'ont les FPGA SRAM à se reconfigurer. De plus, les technologies émergentes telles que la *reconfiguration partielle* [Xili11a] des FPGAs à base de SRAM, de même que la prise en charge de façon native du chiffrement et de l'authentification des bitstreams par certains FPGA [Xili10a] sont des arguments forts qui nous ont confortés dans notre choix.

1.6.2 **Principes de fonctionnement de l'architecture**

Comme cela a déjà été écrit plus haut, le CSS fournit les services cryptographiques et les mécanismes de protection nécessaires au bon fonctionnement d'une radio logicielle sécurisée. Pour une description détaillée du cahier des charges du CSS et de l'impact de celui-ci sur l'architecture du CSS, nous invitons le lecteur à se reporter aux annexes A et B. La Figure 8 illustre l'architecture générale du CSS que nous proposons [GrBG10] et qui fournit les services et mécanismes de communication et de sécurité voulus. On retrouve sur cette figure, le *Security Manager* embarqué au sein d'un *bloc de contrôle* (ContB) et les modules cryptographiques et de bypass qui, eux, sont embarqués dans un *bloc de communication* (ComB).

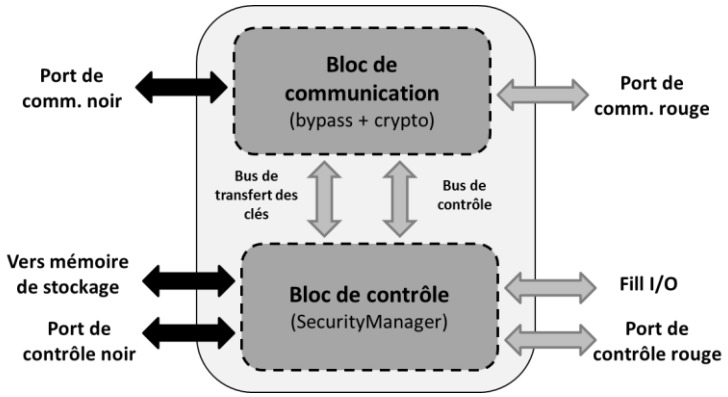


Figure 8: Architecture générale du CSS

Le ContB est contrôlé via les *bus de contrôles rouges et noirs* tandis que les *bus de communication* rouge et noir servent, eux, à transporter les flux de données au travers du ComB. Le ContB contrôle le ComB grâce aux deux bus internes que sont le *bus de transfert des clés* et le *bus*

interne de contrôle. Le bus de transfert des clés permet d'assurer l'isolation des clés et certificats par rapport aux autres types de données manipulés par la radio logicielle alors que, le bus de contrôle permet la transmission de paramètres de configuration (ex : règles de bypass, liste de canaux de communication ouverts). Par ailleurs, le ContB possède aussi la capacité de pouvoir reconfigurer partiellement l'architecture matérielle du ComB. Cette caractéristique est détaillée plus longuement dans les deux sections suivantes. Pour finir, une mémoire de stockage non-volatile (ex : mémoire flash) est utilisée pour le stockage sous forme chiffrée des clés, certificats, algorithmes cryptographiques et logiciels utilisés par la radio logicielle. Cet ensemble de composants qui forme le CSS peut communiquer avec le reste de la radio grâce à quatre modes de communications différents :

- Les modes *rouge/noir* et *noir/rouge* qui permettent le chiffrement et le filtrage des données traversant le CSS.
- Les modes *rouge/rouge* et *noir/noir* qui permettent au CSS de fournir des services cryptographiques aux zones rouges et noires (ex : déchiffrement d'un logiciel chargé en zone rouge).

Les parties suivantes présentent plus en détail le fonctionnement et l'architecture de chacun de ces blocs.

1.6.3 **Le bloc de communication (ComB)**

Le bloc de communication (cf. Figure 9) s'articule autour d'un processeur généraliste et d'un accélérateur cryptographique reconfigurable appelé DRCA pour *Dynamically Reconfigurable Cryptographic Accelerator*. La particularité de ce dernier réside dans sa capacité à pouvoir être partiellement et dynamiquement reconfiguré par le ContB lorsque le besoin s'en fait sentir. C'est-à-dire qu'il est possible de reconfigurer une partie du circuit du ComB sans pour cela interrompre le fonctionnement du reste du circuit. A titre d'exemple, cette capacité peut reposer sur l'utilisation de l'*Internal Reconfiguration Access Port* (ICAP) [Xili10a] embarqué dans les FPGAs produits par la société Xilinx. Par conséquent, il n'est plus nécessaire pour le CSS d'embarquer matériellement l'ensemble des algorithmes cryptographiques qui pourraient potentiellement être utilisés. En effet, une simple reconfiguration du cœur cryptographique permet d'en changer la fonctionnalité sans pour autant avoir à reconfigurer l'ensemble du FPGA implémentant le CSS.

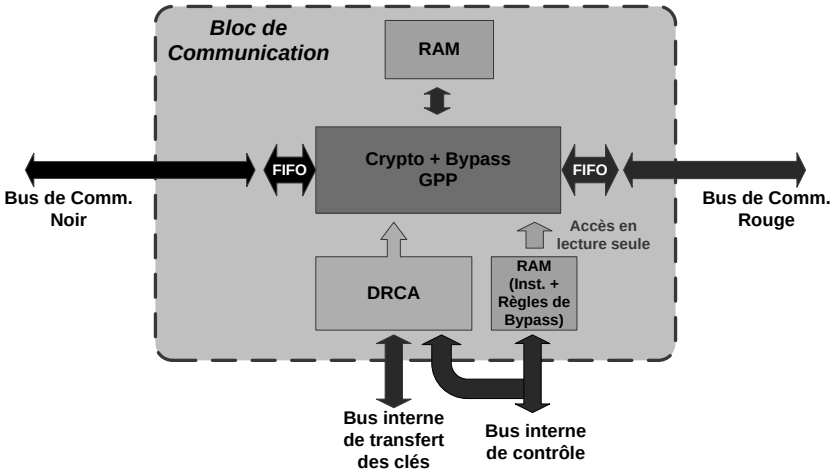


Figure 9: Bloc de communication

C'est donc sur une architecture hybride reposant à la fois sur des composants logiciels (s'exécutant sur le processeur) et des composants matériels (accélérateurs) que repose le ComB. Un tel choix est rendu nécessaire pour deux raisons. D'une part, les solutions hybrides permettent en général d'obtenir un meilleur compromis flexibilité/performance que celles qui ne le sont pas. D'autre part, l'utilisation de composants cryptographiques spécifiques séparés des processeurs généralistes permet d'améliorer la sécurité et la fiabilité globale du système en le partitionnant. La méthode de partitionnement doit donc aussi bien prendre en compte l'aspect performance que l'aspect sécurité. Le supplément sécurité à la SCA apporte déjà un début de réponse quant au partitionnement de la radio. Toutefois, une réponse complète ne peut être apportée qu'en connaissant les spécifications de la radio effectivement développée.

En dehors du partitionnement du ComB, plusieurs moyens de sécurisation supplémentaires peuvent être mis en œuvre. Un moyen d'améliorer la sécurité du système réside dans l'utilisation d'un processeur incluant deux interfaces d'E/S isolées l'une de l'autre : l'une noire, l'autre rouge. En donnant à l'OS du processeur le contrôle des droits en lecture/écriture sur ces interfaces, il est possible de mettre en place un mécanisme simple de contrôle des flux d'information selon le type de l'application exécutée. Par exemple, lors du déchiffrement d'un message, les opérations d'écriture sur l'interface noire et les opérations de lecture sur l'interface rouge sont bloquées. En outre, lors d'un changement de contexte (ex. changement du canal actif, passage du mode de déchiffrement au mode de chiffrement), les zones mémoires partagées entre les différents contextes sont réinitialisées et l'accès aux zones mémoires spécifiques au précédent contexte est bloqué. Pour cela, une *Memory Management Unit* (MMU) sécurisée doit être utilisée comme le demande le supplément sécurité à la SCA. De cette façon, l'isolation et le contrôle de la direction des flux d'information sont assurés.

Un autre moyen de protection consiste à rendre les zones mémoires dédiées au stockage de données non-exécutables. Aujourd'hui la plupart des systèmes d'exploitation modernes (ex. Windows, Linux) embarque des moyens de protection logicielle empêchant par exemple l'exécution de données stockées sur la pile. Bien qu'il soit possible d'envisager l'utilisation de telles solutions logicielles, il reste préférable d'utiliser des solutions strictement matérielles. Le marquage matériel des zones mémoires exécutables (en utilisant les bits mémoire excédentaires de blocs RAM des FPGA) ou plus radicalement l'utilisation de mémoires d'instructions et de données séparées permettent de lutter efficacement contre le détournement des flux d'exécution vers du code malicieux. De la même façon, les règles de filtrage du ComB doivent être stockées dans une mémoire sécurisée et seul le ContB doit y avoir un accès en lecture/écriture dans le but de prévenir toute altération en provenance du ComB. Pour finir, les clés secrètes et certificats cryptographiques sont directement chargés dans le DRCA par le ContB qui n'a qu'un accès en écriture (via le bus de transfert des clés) sur la zone mémoire les contenants. Le ComB n'ayant aucune possibilité d'accéder aux clés et certificats, ceux-ci ne peuvent donc pas fuir directement par les interfaces de communications.

En ce qui concerne la gestion des communications traversant le ComB, habituellement le bus logiciel CORBA utilise le protocole IIOP (*Internet Inter-Orb Protocol*) pour les communications entre les composants d'un système distribué. Malheureusement, IIOP est une spécialisation de GIOP (*General Inter-Orb Protocol*) reposant sur la couche de transport TCP/IP. Or, TCP/IP n'est pas forcément le meilleur protocole de communication entre les composants d'une même radio logicielle. On pourrait préférer, par exemple l'utilisation, de mémoires partagées ou encore d'un bus PCI. Ce problème peut être résolu de deux façons différentes. Soit par l'utilisation d'un protocole basé sur ESIOP (*Environnement Specific-Inter Orb Protocol*) et spécifique à la radio logicielle, soit par l'utilisation de proxys comme cela est expliqué dans un supplément à la SCA [Jtrs07].

La section 1.5.2 apporte plus d'informations sur la manière dont les paquets de données sont traités par le *bloc de communication* du CSS.

1.6.4 **Le bloc de contrôle (ContB)**

Une des caractéristiques principales du CSS est que le ComB est contrôlé par le ContB, ce qui est représenté sur la Figure 10, au travers des interfaces logicielles définies par le supplément sécurité à la SCA. Pour qu'il puisse assurer son rôle, le ContB est construit autour d'un processeur généraliste, le *Security Manager*, aidé dans sa tâche par plusieurs périphériques liés aux interfaces rouges et noires du processeur en fonction de leur rôle et de leur niveau de sécurité. Il est à noter le cas particulier de la mémoire de stockage des clés, celle-ci est connectée à un port spécifique du processeur qui n'est accessible qu'en écriture. Cette mémoire de stockage des clés est chargée en utilisant la procédure suivante : d'abord, les clés et algorithmes cryptographiques sont chargés dans le ContB via un port d'E/S spécifique (Fill I/O) ensuite, ils sont déchiffrés en même temps que leur intégrité et

leur authenticité sont vérifiées par le *Security Manager*. A cet effet, le *Security Manager* utilise un générateur de nombres aléatoires ainsi qu'un accélérateur cryptographique.

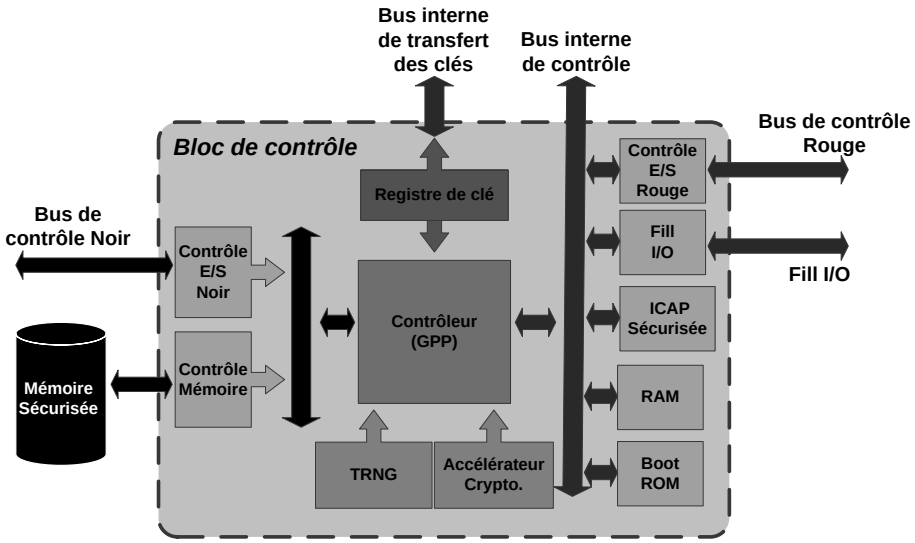


Figure 10: Architecture du bloc de contrôle

En plus des quelques périphériques déjà présentés, le ContB embarque un certain nombre de périphériques. Les périphériques rouges sont les suivants :

- *Un port de contrôle rouge* qui est utilisé pour la réception des instructions de contrôle provenant de la zone rouge.
- *Une ROM de démarrage* qui garantit le stockage et l'intégrité du programme de démarrage du Security Manager.
- *Une RAM interne* au FPGA. Dans le cas où une RAM externe est rendue nécessaire, les données doivent y être stockées sous forme chiffrée et leur intégrité doit être vérifiée.
- *Un ICAP sécurisée* qui permet la reconfiguration partielle du ComB. Cet ICAP est dit sécurisé, car il effectue les reconfigurations partielles dans les limites d'une politique de sécurité qui définit, par exemple, quelles sont les zones qui peuvent être reconfigurées. De plus, l'ICAP vérifie régulièrement l'intégrité du bitstream chargée sur le FPGA afin de prévenir toute altération qu'elle soit intentionnelle (attaque par injection de fautes) ou naturelle (*Single Event Upset*).

Quant aux périphériques noirs, ils sont au nombre de deux :

- *Un port de contrôle noir* qui est utilisé pour la réception des instructions de contrôle provenant de la zone noire.

- *Un contrôleur mémoire sécurisé* qui est utilisé pour stocker de façon permanente des données ou programmes rouges dans une mémoire non volatile. (ex : mémoire flash).

Comme cela a été écrit, le *Security Manager* et ses périphériques servent à contrôler la configuration du ComB et notamment à gérer les *canaux de communication* ouverts. Un *canal* sert à encapsuler un flux de communication spécifique qui traverse le ComB. On peut distinguer deux types de canaux, les *canaux systèmes* et les *canaux utilisateurs*. Seulement deux canaux systèmes sont actifs à chaque instant, chacun d'eux est utilisé pour transférer des messages de Type 1 et de Type 2. Ceux-ci sont initialisés au démarrage du système d'après les règles de filtrage spécifiques à ces types de message. A l'inverse, un nombre quelconque de canaux utilisateurs peuvent être instanciés à chaque instant. Chaque canal est défini par les paramètres suivants :

- Un mode (r/r, b/b, simplex, half-duplex, full-duplex)
- Un protocole cryptographique
- Une ou des clés de chiffrement pour les algorithmes symétriques
- Un ou des certificats
- Plusieurs règles de filtrage

Lorsqu'un nouveau canal est ouvert, le ContB lui attribue un identifiant puis, il charge les règles de filtrage correspondantes dans la mémoire du ComB. Cet identifiant sera par la suite utilisé pour marquer les messages qui devront emprunter ce canal de manière à ce que le ComB puisse y appliquer les bons algorithmes et règles de filtrage.

1.6.5 **Fonctionnement du CSS : Exemples d'illustration**

Dans cette partie, le fonctionnement du CSS est présenté au travers de trois cas de figure : *l'enregistrement d'une clé cryptographique* dans le CSS, *l'ouverture d'un canal de communication* et *le chiffrement d'un paquet de données*.

Enregistrement d'une clé cryptographique

Etant donné qu'un FPGA SRAM ne permet pas de stocker de façon permanente et sécurisée les clés chiffrement (pas de mémoire non volatile), des supports de stockages spécialisés doivent alors être utilisés et les clés doivent être chargées dans le CSS à chaque mise sous tension de la plateforme radio. Voici les étapes de ce processus :

1. L'utilisateur muni des autorisations nécessaires lance la procédure de chargement des clés cryptographiques.
2. Le processeur généraliste du *bloc de contrôle* entre dans un *mode de fonctionnement sécurisé* chargé d'exécuter un code spécifique dans un espace mémoire sécurisé. Il s'isole de l'ensemble de ses périphériques à l'exception du port de chargement des clés cryptographiques (Fill I/O) et de ces périphériques de déchiffrement.

3. Les clés sont lues sur le port d'entrée, déchiffrées puis transférées dans la mémoire de clé du *bloc de communication*.
4. Une fois les transferts de clés effectués, le processeur sort de son mode de fonctionnement sécurisé ce qui provoque la remise à zéro de l'ensemble des zones mémoires allouées à ce mode de fonctionnement. La main est finalement rendue à l'utilisateur.

Ouverture d'un canal de communication

L'ouverture d'un canal de communication sur le CSS suit les étapes suivantes :

1. L'utilisateur muni des autorisations adéquates lance la procédure d'ouverture d'un nouveau canal de communication en spécifiant l'ensemble des paramètres de configuration relatifs à cette action.
2. Le *bloc de contrôle* vérifie au préalable que la configuration du canal est valide (ex. niveau de sécurité de la clé cryptographique adapté aux données à chiffrer). Il vérifie ensuite que les ressources matérielles disponibles sont suffisantes pour permettre l'ouverture d'un nouveau canal. Finalement, les données de configuration du canal sont sauvegardées en mémoire.
3. Un identifiant de canal est alloué au canal nouvellement créé.
4. Le *bloc de contrôle* notifie l'ouverture du nouveau canal au processeur généraliste et au DRCA du *bloc de communication* et leur transmettant les informations nécessaires (ex. règles de filtrage, type de chiffrement à appliquer, etc.).
5. Si l'exécution des étapes précédentes n'a conduit à aucune erreur, la main est rendue à l'utilisateur et l'identifiant du canal lui est transmis. Sinon, la cause de l'erreur lui est notifiée et une alerte est éventuellement levée.

Traitement d'un message par le bloc de communication

En faisant abstraction de la couche de transport utilisée et sans perte de généralité, le processus de traitement d'un message entrant peut être subdivisé en plusieurs étapes. Par exemple, dans le cas d'une transmission depuis la zone rouge vers la zone noire, ces étapes consistent à :

1. *Extraction du message :* Le message est extrait du protocole de transport.
2. *Analyse du message :* L'en-tête et le corps du message sont séparés.
3. *Validation de l'en-tête :* L'en-tête est validé en utilisant les règles de filtrage courantes. Dans le cas où l'en-tête ne satisfait pas aux règles établies, le message est rejeté et le canal de communication est automatiquement fermé.
4. *Chiffrement ou validation du corps de message :* Si le message est de Type 3 et que l'en-tête a été validé, alors le message est chiffré par le CSS. Par contre, si le message est de Type 1 ou 2 alors les règles de filtrage sont aussi appliquées à son corps.
5. *Encapsulation de message :* Une fois le message validé et chiffré, il est à nouveau encapsulé avec le protocole de la couche de transport afin d'être envoyé.

Pour les transferts rouge/noir, rouge/rouge et noir/noir, seule la 4ème étape nécessite d'être modifiée.

1.7 **Synthèse**

Dans le cadre de la radio logicielle, lorsqu'on excepte les attaques matérielles nécessitant la proximité de l'attaquant avec le matériel à attaquer, il reste tout un éventail d'attaques basées sur les deux grandes catégories suivantes:

- Les attaques visant les signaux transmis entre plusieurs radios.
- Les attaques visant des failles de conception des composants constituants les plateformes radios.

Ces deux grands types d'attaques peuvent être prévenus dans une large mesure par deux mécanismes qui sont :

- La mise en œuvre de protocoles cryptographiques permettant la protection des flux d'informations.
- L'isolation fonctionnelle de chacune des parties constitutives d'une radio logicielle sécurisée.

Le CSS tel que défini par le supplément sécurité à la SCA vise la fourniture de ces deux mécanismes en permettant la mise en œuvre d'une architecture en coupure rouge/noir permettant à la fois la sécurisation des flux d'informations et l'isolation fonctionnelle des composants d'une plateforme radio. Le CSS lui-même repose sur le principe d'isolation fonctionnelle dans la mesure où chacune de ses sous-parties (module de bypass, unité(s) cryptographique(s) et Security Manager) est isolée des autres.

L'architecture théorique du CSS reconfigurable proposée dans cette partie est conçue de manière à répondre à l'ensemble des points du cahier des charges du CSS tel que défini par le supplément sécurité à la SCA. Pour cela, le CSS en découpé en deux parties : le *bloc de contrôle* et le *bloc de communication*. Le premier se charge exclusivement de la gestion de la configuration du CSS tandis que le second se charge exclusivement du chiffrement/déchiffrement et du filtrage des flux d'informations. En outre, le DRCA du *bloc de communication* est reconfigurable de manière à permettre une compatibilité étendue aux futurs algorithmes et protocoles cryptographiques. Ce type de fonctionnement permet de garantir l'application du principe d'isolation. On remarque qu'en outre chaque partie du CSS a un accès limité aux seules informations qui la concerne. Le bloc de contrôle n'a par exemple pas accès au flux d'informations traités par le bloc de communication. Pareillement, le processeur généraliste du *bloc de communication* n'a pas non plus l'accès aux clés de chiffrement, seul le DRCA y a accès.

Notre architecture théorique prend seulement en compte les problématiques relatives à la partie matérielle du CSS. Les aspects logiciels n'ont donc pas été abordés dans la

présentation de cette architecture. Le lecteur trouvera toutefois dans les annexes A et B une description complète de l'ensemble des points du cahier des charges du CSS et de la manière dont ils doivent être implantés (sous forme matérielle et/ou logicielle).

Pour finir, les parties précédentes ont montré que la structure du CSS est principalement basée sur des composants communs dont l'étude et/ou la conception ne revêt qu'un intérêt limité. Toutefois, le DRCA fait exception à cette constatation dans la mesure où la conception de crypto-systèmes efficients est une problématique complexe. Elle fait d'ailleurs, comme nous le verrons par la suite, toujours l'objet de travaux de recherche et ce d'autant plus que l'on souhaite tirer profit des capacités de reconfiguration des FPGA.

1.8 **Conclusion**

En conclusion, ce chapitre a montré en quoi l'évolution de services de télécommunication a été à l'origine de l'apparition du concept de radio logicielle. On retiendra que la principale valeur ajoutée de la radio logicielle par rapport à la radio classique se situe au niveau de la flexibilité et des coûts de fabrication. Néanmoins, il reste encore un certain nombre de verrous technologiques qui n'ont pu être totalement levés, notamment en ce qui concerne les performances en termes de débit. Pour pallier à cela, la conception des radios logicielles adopte le plus souvent une approche hybride alliant composants logiciels et accélérateurs matériels.

Un autre problème réside dans la nécessité de protéger à la fois les données échangées et les matériels qui les échangent. Par ailleurs, on peut remarquer qu'une radio logicielle est conçue de manière à être flexible et à évoluer au cours du temps. Il s'ensuit qu'une telle radio est particulièrement sensible aux attaques logicielles. A cela s'ajoute le fait que les mécanismes utilisés pour protéger l'information nécessitent le plus souvent une importante puissance de calcul. En réponse au premier problème, le gouvernement américain a publié en 2004 un supplément sécurité à la radio logicielle qui formalise la conception d'une architecture de radio logicielle sécurisée. Par contre, le second problème reste entier dans le sens où l'utilisation d'accélérateurs matériels pour les applications cryptographiques s'oppose de fait au concept de radio logicielle. Cette affirmation est toutefois à nuancer dans la mesure où la mise en œuvre d'accélérateur matériel implantant les *standards cryptographiques* actuels n'est pas incompatible avec la philosophie de la radio logicielle *restreinte*. Quoiqu'il en soit, la dernière partie de ce chapitre apporte une réponse générale à ce problème en introduisant la notion de *Crypto Sub System* partiellement reconfigurable. De cette façon, bien que les algorithmes cryptographiques soient implémentés sous forme matérielle, cela ne nuit pas à la flexibilité de la radio.

Cependant, en l'état, l'étude n'est que partielle. En effet, malgré le fait que l'architecture proposée apporte certaines réponses aux problématiques de sécurité des radios logicielles, elle n'explique pas comment les primitives cryptographiques, qui sont les pierres angulaires d'une radio sécurisée, sont introduites au sein d'un CSS. La deuxième moitié de ce document essaye d'apporter une réponse à ce problème. Mais, avant cela, il s'agit de

présenter les travaux qui existent concernant l'implantation de primitives cryptographiques. L'objectif étant de savoir dans quelle mesure ces implantations sont adaptées ou non à la radio logicielle.

Chapitre 2

Etat de l'art des architectures matérielles de crypto-systèmes

2.1 Introduction

Le développement des technologies de l'information et des communications est à l'origine de la démocratisation de la cryptographie. En effet, les techniques de chiffrement et d'authentification de l'information sont couramment utilisées dans de nombreuses applications telles que le paiement sur internet, les réseaux privés virtuels ou le contrôle d'accès aux contenus soumis au droit d'auteur. C'est ainsi que les services cryptographiques standards que sont le chiffrement, l'authentification, l'intégrité et la non-répudiation sont devenus des fonctionnalités incontournables de nombreux systèmes de traitement de l'information. Pour faire face à cette nouvelle demande, les concepteurs de systèmes se sont basés sur des standards mondialement reconnus (ex. DES, MD5, RC4) dont la sécurité est maintenant contestée [FlMS01]. En effet, sécurité et performance (en termes de coût, débit, latence et consommation d'énergie) sont des notions qui tendent à s'opposer [LeKS10, ZSFZ11]. Or, la qualité de certains services (ex : VoIP) dépend fortement des performances de l'infrastructure de communication qu'ils utilisent [RaTa10]. Il s'ensuit que, parfois, le niveau de sécurité effectif d'un système ou d'un protocole n'est pas celui qu'il aurait dû être (ex : cas du protocole WEP). Pour finir, la multiplication des protocoles de communication et de chiffrement rend ardu la mise au point de systèmes interopérables. En résumé, lors de la conception d'un système sécurisé, un concepteur doit prendre en compte les aspects principaux suivants : la *sécurité*, les *performances* (en termes de consommation d'énergie, de débit ou encore d'encombrement), la *flexibilité* et le *coût*. Chacun de ces aspects ayant un poids plus ou moins important selon le marché ciblé. Par exemple, pour les applications militaires, l'aspect sécurité est primordial alors que pour les entreprises spécialisées dans les télécommunications les performances ne peuvent être négligées. En ce qui concerne la radio logicielle, c'est l'aspect flexibilité qui ne peut être ignoré.

Comme cela a été expliqué, l'ajout de mécanismes de sécurité a généralement un impact négatif sur les performances (débit et surface silicium occupée) d'un système. Il existe aujourd'hui un grand nombre de travaux traitant de l'implantation matérielle d'algorithmes cryptographiques dans le but d'en améliorer les performances. D'ailleurs, il est intéressant de voir par quels moyens ces circuits répondent aux problématiques précédentes, c'est l'objet de ce chapitre. Toutefois, nous limiterons cette étude aux problématiques de sécurité, de performance (en termes de débit et latence) et de flexibilité et nous ferons

l'impasse sur la problématique du coût financier qui n'est pas forcément pertinente dans le cadre de recherches amont. Par ailleurs, ce document se focalise uniquement sur le sujet des algorithmes de chiffrement symétriques et le sujet des algorithmes de chiffrement asymétriques n'y est pas abordé. En effet, les premiers sont préférés aux seconds lorsque l'on souhaite chiffrer des flux de données.

Cette partie se décompose de la manière suivante : dans un premier temps, le compromis entre performance, flexibilité et sécurité est étudié (section 2.2). Cette étude sert de base à la classification des différents types d'architectures dédiées à la cryptographie qui est faite dans un second temps (section 2.3) et qui met l'accent sur un certain nombre d'architectures dont les propriétés sont intéressantes dans le cadre de la radio logicielle. Par la suite, ces architectures font l'objet d'une étude plus poussée (section 2.4) qui permet de conclure quant à leur adéquation avec les besoins de la radio logicielle (section 2.4.6) et sur les mesures qui doivent être prises afin de concevoir une architecture efficiente dans le domaine de la cryptographie pour la radio logicielle sécurisée (section 2.5).

2.2 La radio logicielle et le compromis performance / flexibilité / sécurité

Un composant cryptographique peut être conçu sur la base d'une des nombreuses architectures existantes. L'ensemble de ces architectures forme ce que l'on appelle un espace de conception dont les dimensions correspondent à des caractéristiques générales telles que les performances (en termes de débit et latence), la flexibilité, la sécurité, les coûts financiers, … A un instant donné, cet espace est borné par les technologies disponibles. Chaque application nécessite une combinaison différente des précédentes caractéristiques qui correspond à une partie de l'espace de conception et donc un ensemble d'architectures qui évolue en fonction des technologies disponibles. Comme cela a été expliqué précédemment, le concept de la radio logicielle privilégie la flexibilité aux performances. Quant à l'aspect sécurité, les applications militaires auront tendances à se situer dans la moitié de l'espace qui englobe les architectures les plus sécurisées, tandis que les applications civiles se situeraient plutôt dans la moitié restante de l'espace de conception.

Jusqu'au début des années 2000, l'espace de conception des circuits électroniques était restreint. En effet, seules deux solutions étaient alors envisageables : les processeurs généralistes (GPP) qui sont flexibles mais peu performants étant donné leur mode de fonctionnement séquentiel et les circuits intégrés dédiés à une application (ASIC) performants mais peu ou pas flexibles. Afin de pallier aux limitations de ces technologies, des architectures hybrides ont vu le jour. Celles-ci réunissent sur une même puce, un processeur généraliste programmable et un circuit électronique dédié implémenté sous la forme d'un accélérateur matériel connecté au processeur.

Ce type d'architecture aujourd'hui largement utilisé pose toutefois un problème. En effet, dans le cadre de la radio logicielle, les composants doivent être génériques et programmables. Ce qui peut potentiellement obliger à embarquer un grand nombre

d'accélérateurs afin d'anticiper les besoins. Encore faut-il que cette anticipation soit possible… Par ailleurs, l'apparition de techniques d'attaques telles que les attaques par canaux cachés nécessitent la mise en œuvre d'un certain nombre de contremesures matérielles. Or, de nouvelles attaques peuvent apparaître tout au long du cycle de vie d'un appareil, les composants matériels ayant une structure figée (ex : ASIC, GPP, DSP, …) sont donc en principe moins sûrs.

A partir des années 2000, l'utilisation de composants basés sur une logique programmable (ex : FPGA, CPLD, EPLD) s'est généralisée. Ceux-ci ont largement évolués ces dernières années au point que les tout derniers FPGA SRAM Virtex 7 produits par Xilinx embarquent jusqu'à 305 400 éléments logiques programmables chacun d'eux étant constitué de quatre LUT à six entrées ainsi que huit bascules. La faible granularité des FPGA et leur haut degré de parallélisme sont des atouts pour un bon nombre d'application et notamment les applications cryptographiques [WoGP04, WoPa03]. A cela s'ajoute le fait que les FPGA sont principalement basés sur des technologies FLASH ou SRAM (environ 90% du marché) qui leur permettent d'être configurés et reconfigurés et donc de proposer des fonctionnalités qui évoluent au cours du temps par le biais de mises à jour matérielles. De cette façon un algorithme peut être remplacé par un autre et de nouvelles contremesures matérielles peuvent être ajoutées. Ce type de composant est aujourd'hui largement utilisé dans les systèmes à base de radio logicielle. Par exemple la radio SPEAKeasy a été l'un des premiers systèmes à utiliser des FPGA dans le but de traiter les signaux radios [CoBo99]. Une autre solution permettant d'améliorer la flexibilité des composants cryptographiques par rapport aux architectures à base d'accélérateurs *cryptographiques* dédiés réside dans l'utilisation de processeurs possédant un jeu d'instructions adapté à la cryptographie. Ce genre d'approche se prête bien à une implantation sur des puces reconfigurables telles que les FPGA [GaFB11]. Les derniers développements concernant ces architectures portent sur la mise au point d'architectures multicœur programmables et reconfigurables. Le reste de ce chapitre reviendra plus longuement sur cette approche.

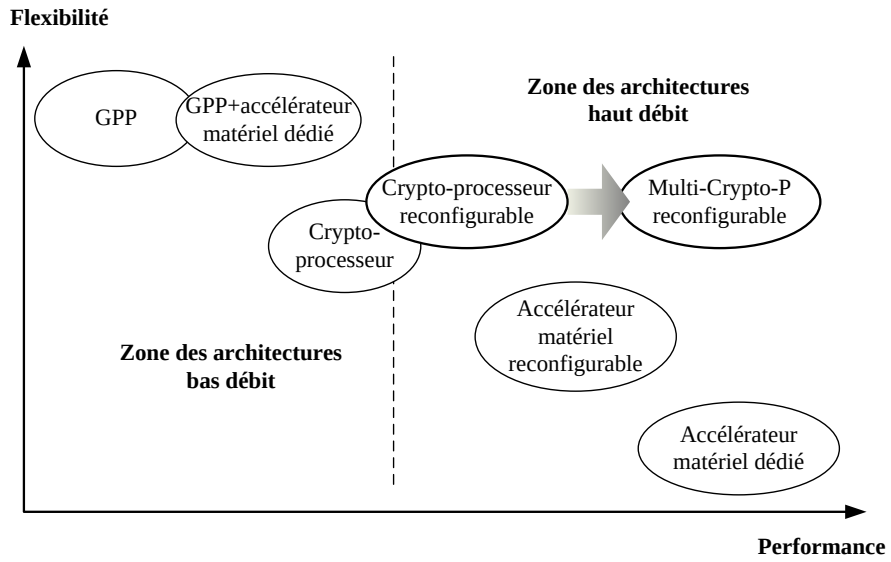


Figure 11: Espace de conception (performance/flexibilité) des composants cryptographiques

La Figure 11 présente une projection en deux dimensions dans le plan performance/flexibilité de l'espace de conception des composants cryptographiques. L'axe des abscisses représente les performances de l'architecture en termes de puissance de calcul. Il englobe donc des notions telles que le débit ou la latence d'une architecture. L'axe des ordonnées représente quant à lui la flexibilité, c'est-à-dire la capacité des architectures à exécuter des algorithmes différents. Le GPP est l'architecture la plus flexible dans le sens où changer le service fourni par un GPP revient à changer le programme qu'il exécute. Par ailleurs, la position des différents types d'architectures dans l'espace de conception montre que la flexibilité et les performances sont des caractéristiques antagonistes. Toutefois, les architectures à base de cryptoprocesseur à la fois multicœur et reconfigurable améliorent le compromis performance/flexibilité, ce qui correspond aux besoins de la radio logicielle. Si la dimension représentant la sécurité n'apparait pas dans cette figure c'est parce que, à notre connaissance, il n'existe aucune étude prouvant la supériorité en terme de sécurité d'une architecture par rapport à une autre. En outre, il semble que les axes de recherche que sont la sécurité et les performances sont dans les faits disjoints. En fait, la littérature manque de travaux abordant de manière globale les aspects sécurité et performance. Or, même si l'étude du niveau de sécurité intrinsèque des architectures numériques est un sujet pertinent, celui-ci déborde manifestement du cadre de cette thèse.

Nous venons d'expliquer que chaque type d'architecture définit une région de l'espace de conception. De plus, à chaque application caractérisée par un certain nombre de besoins correspond une région de l'espace de conception. Il s'agit donc de déterminer la région de l'espace de conception occupée par les composants cryptographiques destinés à la radio logicielle en précisant leurs besoins. De façon succincte, on peut citer :

- *Les communications multicanal :* Le composant cryptographique doit pouvoir gérer plusieurs canaux en même temps, il doit donc permettre le passage rapide d'un canal à un autre. Il doit aussi garantir l'isolation entre les canaux que ce soit au niveau des clés de chiffrement utilisées ou des données traitées.
- *L'interopérabilité :* le composant cryptographique doit pouvoir supporter différents types de protocole cryptographique et différents niveaux de sécurité afin d'être compatible avec un large nombre de standards de communication.
- *L'évolutivité :* Le composant doit être évolutif afin de préserver l'interopérabilité dans le temps et dans le but de protéger la radio en mettant à jour les mécanismes de contremesure.
- *Les performances :* Le composant doit garantir des performances en termes de débit et latence suffisantes pour les applications de transfert de fichier, de VoIP et éventuellement de vidéo temps réel et de contrôle temps réel.
- *La sécurité des données :* Un composant cryptographique doit bien entendu garantir la sécurité de l'ensemble des données dont il a la charge. C'est-à-dire, de façon non exhaustive, les données à chiffrer ou déchiffrer, les clés et certificats cryptographiques, les paramètres de configurations.

Chacun de ces besoins contraint la région de l'espace de conception qui doit être occupée par l'architecture d'un composant cryptographique utilisé dans une radio logicielle sécurisée. Nous invitons le lecteur à se reporter aux annexes A et B pour avoir une vision plus précise du cahier des charges du CSS et de son influence sur les composants cryptographiques intégrés au CSS. La partie suivante présente plusieurs architectures tirées de la littérature qui forment l'espace de conception des composants cryptographiques. Elle se terminera par une synthèse résumant les points forts et les points faibles de chacune des architectures présentées ainsi que leur adéquation avec le principe de la radio logicielle.

2.3 Classification des architectures matérielles de crypto-systèmes

Il existe un grand nombre de travaux traitant des architectures cryptographiques et chacun d'eux utilise des terminologies différentes. Il s'ensuit que cet ensemble manque en apparence de cohérence. Dans un souci de clarté, on distinguera donc : les *architectures programmables* qui exécutent un *programme* basé sur un jeu d'instructions, les *architectures reconfigurables* dont la structure (ex : chemin de données) peut être modifiée en fonction des besoins et les *architectures dédiées* qui bien qu'elles puissent être configurables (ex : sélection d'une taille de clé) ont une structure figée qui n'est pas programmable. La suite de cette partie présente les caractéristiques principales de chacune de ces classes d'architecture avant de les illustrer par quelques exemples tirés de la littérature.

2.3.1 Architectures programmables

En ce qui concerne les architectures programmables, on peut distinguer deux cas de figure. Le premier cas consiste à modifier le jeu d'instructions d'un processeur généraliste afin d'y ajouter des instructions spécifiques à la cryptographie comme l'a fait Intel avec ses

processeur Core i5 [Guer10]. Le second cas consiste à créer un cryptoprocesseur avec un jeu d'instructions ad-hoc et une architecture définie en fonction des besoins. Le contrôle des ressources via un jeu d'instructions implique de facto que les ressources cryptographiques font parties intégrantes du chemin de donnéess du processeur.

La Figure 12.a présente l'architecture globale d'un processeur généraliste modifié dans le but de fournir des services cryptographiques. Ici, la modification consiste à insérer dans le processeur une ou plusieurs ALU embarquant des opérateurs spécifiques à la cryptographie tels que des tables de substitution ou des multiplieurs dans les corps de Galois. Ces opérateurs sont accessibles au travers du jeu d'instructions du processeur. Les données confidentielles comme les clés secrètes sont stockées dans la mémoire de donnée du processeur au même titre que n'importe qu'elle autre donnée. La principale difficulté rencontrée lors de la conception de ce type d'architecture consiste à la mise au point du jeu d'instructions dédié à la cryptographie (notamment son niveau de granularité) et éventuellement du compilateur qui va avec. Dans [RRPS02], Ravi et al. proposent un flot de conception destiné aux processeurs dédiés aux applications cryptographiques. En modifiant un processeur 32 bits Xtensa de Tensilica, ils sont arrivés à montrer qu'une implantation à base de processeur généraliste personnalisé pouvait fournir des performances bien supérieures à celles d'une implantation basée sur un processeur généraliste non-personnalisé. Cette méthode de co-design fut le point de départ de nombreux travaux de recherche destinés aux architectures parallèles [SBPV06] et aux systèmes embarqués [ScVe03].

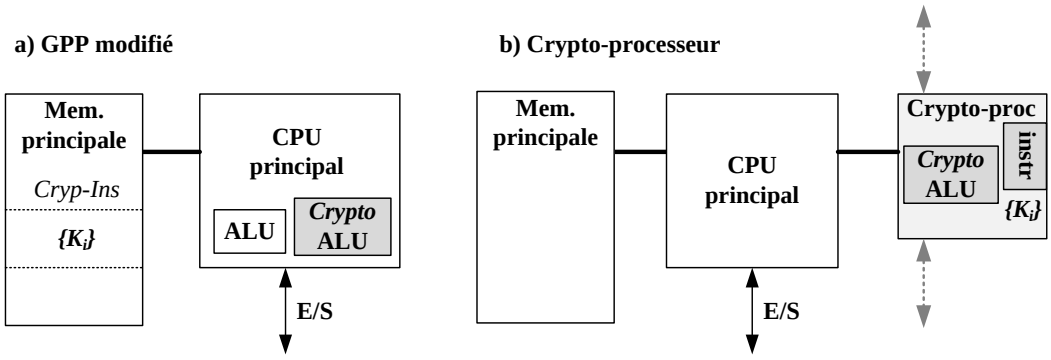


Figure 12: Crypto-systèmes à architectures programmables

Plusieurs travaux se sont concentrés sur la mise au point d'extensions dédiées à AES pour les jeux d'instructions des processeurs généralistes [HäHH07]. A l'origine, ces travaux ce sont principalement portés sur la mise au point d'instructions dédiées aux opérations de substitution (S-Box) dont les implantations logicielles sur GPP sont peu performantes [BuMA00, TiGS00]. Par la suite, Tillich et al. ont développé plusieurs jeux d'instructions destinés aux processeurs RISC 32 bits (ex : MIPS, ARM, SPARC) permettant l'accélération d'AES dans son ensemble [TiGr06, TiHe08]. En ce qui concerne les produits

commerciaux, Intel a doté ses processeurs Intel Core i5 et i7 de six instructions permettant l'accélération matérielle des opérations de chiffrement basées sur AES [Guer10]. Elles fournissent le support nécessaire aux opérations de chiffrement, de déchiffrement et de génération des clés de round. En outre, elles sont conçues de manière à prévenir les attaques par canaux cachés du type *timing attack* ou *cache attack*. Un autre exemple intéressant est fourni par Xilinx et son projet *CryptoBlaze* [Xili03] basé sur le contrôleur 8 bits *Picoblaze* du même constructeur. Dans ce projet, un multiplieur de Galois est ajouté au chemin de donnéess du *Picoblaze* (cf. Figure 13) dans le but d'accélérer les algorithmes cryptographiques qui utilise ce type d'opérateur (ex : AES). Une mémoire faisant office de S-Box est aussi accolée au microcontrôleur afin de faciliter les opérations de substitution. L'avantage de ce genre d'architectures est leur autonomie. En effet, le processeur original a été conçu afin d'être utilisé de façon autonome (cf. processeur Intel) contrairement aux ressources cryptographiques qui nécessitent la présence d'un contrôleur principal. Le défaut de ce type d'architectures provient du fait qu'à l'origine celles-ci n'ont pas été pensées pour effectuer des calculs cryptographiques. Leur structure interne (banc de registre, chemin de donnéess, …) n'est donc pas toujours adaptée aux besoins des applications cryptographiques en termes de sécurité ou de performance.

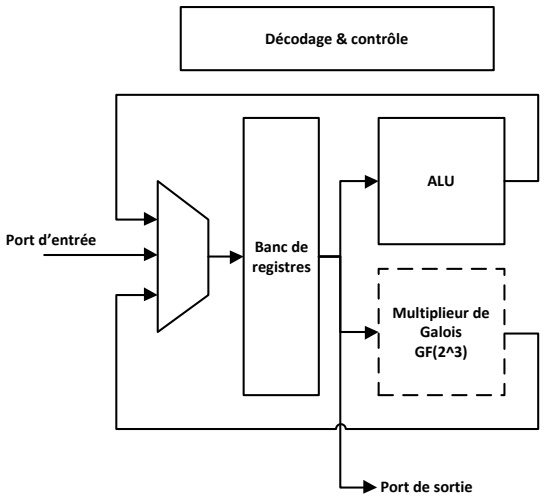


Figure 13 : Schéma simplifié du chemin de données du *Cryptoblaze*

Afin de palier à ce problème des processeurs ayant un jeu d'instructions *ad-hoc* ont vu le jour. La Figure 12.b présente l'architecture générale d'un processeur dédié à la cryptographie encore appelé *Cryptoprocesseur*. Ce genre de processeur, qui est nécessairement couplé à un processeur généraliste, embarque une ALU dédiée à la cryptographie. Son jeu d'instructions est donc principalement composé d'instructions dédiées aux calculs cryptographiques. Le niveau de granularité des opérateurs embarqués dépend des besoins de l'application. On pourra aussi bien retrouver des opérateurs de bas niveau (ex : multiplieurs de Galois, S-Box, …) que des opérateurs de haut niveau (ex : opérateur

effectuant un calcul d'AES complet). De telles architectures sont plus légères, plus flexibles et plus performantes que les architectures basées sur des processeurs généralistes modifiés. Toutefois, l'aspect sécurité n'est pas systématiquement pris en compte lors de leur conception. Les cryptoprocesseurs sont utilisés dans de nombreuses applications et notamment dans la radio logicielle [MaNM08] en tant que sous système cryptographique comme l'a montré le premier chapitre.

Les architectures des cryptoprocesseurs ont fait l'objet de différents travaux portant notamment sur la mise au point d'architecture du type *Very Long Instruction Word* (VLIW) destinée au chiffrement symétrique et au hachage. En effet, les algorithmes de chiffrement symétrique (ou de hachage) utilisent des tailles de blocs allant de 64 à 512 bits mais travaillent en réalité sur des mots de 8, 16 ou 32 bits voir 64 pour les algorithmes de hachage. Le processeur *Cryptonite* est un bon exemple d'architecture VLIW embarquant deux chemins de données sur 64 bits [BuHO04]. Celui-ci supporte les algorithmes AES, DES et 3DES ainsi que MD5. Pour cela, il embarque des instructions permettant d'effectuer des *permutations de bit,* des *rotations* et des *xor*, les opérations de substitution étant effectuées grâce à des mémoires dédiées. Le projet *Cryptomaniac,* détaillé plus loin dans ce chapitre, est un autre exemple d'architecture VLIW dédiée à la cryptographie. Cette architecture est capable d'exécuter jusqu'à quatre instructions sur 32 bits par cycle d'horloge [WKWA01, WuWA02]. En outre, grâce à l'utilisation d'instructions admettant trois opérandes en entrée, cette architecture permet de combiner plusieurs instructions ayant une faible latence afin de les exécuter en un seul cycle. Plus récemment, Theodoropoulos et al. [ThPP08, ThSP09] ont présenté des travaux sur un cryptoprocesseur compatible avec de nombreux algorithmes de chiffrement tels qu'AES, Twofish, Serpent ou encore RC4. Parallèlement à ces travaux, il existe de nombreux cryptoprocesseurs commerciaux et nous citerons par exemple le *NEC MP211* dédié au marché des systèmes embarqués mobiles [ARRS06] et à celui des *Trusted Platform Module* (TPM) [Tcpa03] qui sont utilisés dans les ordinateurs portables.

2.3.2 **Architectures à base de coprocesseurs dédiés**

Lorsque les architectures *programmables* ne permettent pas d'atteindre les performances escomptées, le concepteur a la possibilité d'utiliser des architectures *dédiées* (cf. Figure 14). Ici, chaque coprocesseur est dédié à un algorithme particulier. Le caractère spécifique de ces unités de calcul fait qu'elles ont un ration débit/surface occupée très intéressant, que ce soit pour les architectures haut débit [HoVe06, LWFB07] ou les architectures compactes [AzIk07, ChGa00, LoRD06]. Toutefois, ces très bonnes performances sont contrebalancées par le surcoût induit par les communications entre le processeur et ses coprocesseurs. Les travaux menés par Hodjat et al. illustrent de façon notable ce fait. Dans [HoVe04], Hodjat et al. présentent un système cryptographique composé d'un processeur LEON de type SPARC V8 épaulé par un coprocesseur cryptographique dédié à AES. Avec cette architecture, le processus de chiffrement d'un bloc de 128 bits prend 704 cycles alors que seulement onze cycles sont réellement passés à chiffrer le bloc de donnée. Les cycles

restant étant consommés par les mécanismes de communication entre le processeur et le bloc AES.

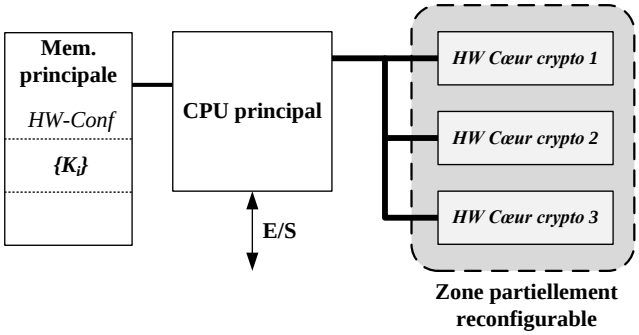


Figure 14: Crypto-système à coprocesseurs dédiés reconfigurables

Par la suite, d'autres architectures plus évoluées ont vu le jour et notamment des architectures basées sur des structures multicœur proches de l'illustration de la Figure 14. Parmi celles-ci on retrouve un coprocesseur multicœur [WSHW10] basé sur le coprocesseur mono-cœur *ASTHETIC* [CCCC05]. Cette architecture multicœur, présentée en détail plus loin dans ce chapitre, permet d'atteindre des débits de plusieurs gigabits par seconde sans pour autant être limité par des problèmes de bande passante au niveau des bus d'E/S. Deux autres exemples de coprocesseurs AES multicœur contrôlés par un processeur *MOLEN* ont été présentés dans [CKVS06, PCGV08]. La première architecture citée est composée d'un processeur généraliste et de plusieurs unités reconfigurables pour les calculs cryptographiques (*CrCU*) qui sont interconnectées en réseau. Chaque *CrCU* embarque un cœur AES complet, un registre de clé et une unité de contrôle. La seconde architecture est destinée au chiffrement en parallèle de plusieurs flux de données (*Multi-Stream AES*). Elle embarque des cœurs AES et une unité de contrôle. Cette dernière gère l'activation des cœurs et la configuration des multiplexeurs qui permettent le transfert des données à chacun des cœurs AES. On peut aussi citer l'architecture *CryptoBooster* qui bien que plus ancienne (1999) ciblait déjà la technologie FPGA [MTRG99]. Bien que la structure d'un coprocesseur dédié soit fixe, l'utilisation de la technologie FPGA et notamment de la reconfiguration partielle permet d'en améliorer la flexibilité. La Figure 14 montre que les coprocesseurs cryptographiques peuvent être embarqués dans une zone reconfigurable de manière à ce que l'algorithme qu'ils implantent puisse être remplacé par d'autres si le besoin s'en faisait sentir. Toutefois, un grand nombre de puces repose sur des technologies ASIC et ont donc une structure figée qui interdit toute modification des fonctionnalités.

2.3.3 **Architectures à base de coprocesseurs reconfigurables**

Contrairement aux coprocesseurs *dédiés*, les coprocesseurs *reconfigurables* ont un chemin de données reconfigurable ce qui les rend plus flexibles. Celui-ci varie au cours du temps en

fonction du type de traitement appliqué à une donnée. La Figure 15 illustre la structure d'un coprocesseur basé sur une architecture matérielle reconfigurable à gros grain. Depuis le début des années 2000, les architectures reconfigurables occupent une place de plus en plus importante dans le monde des architectures matérielles numériques. Plusieurs travaux ont démontré l'efficacité de ce type d'architecture en ce qui concerne le calcul intensif [TrSh03], les systèmes embarqués [GCSB06] ou la cryptographie [MVCT07]. En étudiant l'espace de conception des architectures reconfigurables, Bossuet et al. [BoGP05] ont montré l'intérêt que peut avoir ce type d'architecture dans les domaines du traitement du signal ou du chiffrement symétrique. De façon plus concrète, il est possible de citer au moins trois projets dans le domaine de la cryptographie qui tirent profit de ce type d'architecture : *Celator* [FrPP08], *COBRA* [ElPa03] et *CryptArray* [Lomo04]. De façon générale, ces architectures sont composées d'unités de calcul gros grain reliées entre elles par un réseau d'interconnexions, le tout étant conçu pour être reconfigurable. Ces architectures font l'objet d'une étude plus détaillée dans la suite de ce chapitre.

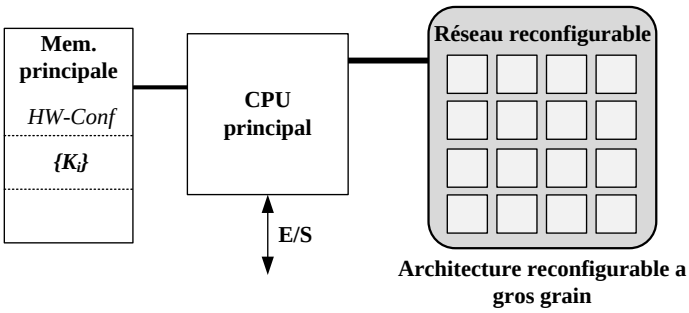


Figure 15: Coprocesseur reconfigurable à gros grain

Parallèlement aux travaux menés sur les architectures à gros grains, d'autres portants sur les architectures à grains fins ont vu le jour. Wollinger et al. ont montré dans [WoGP04, WoPa03] les bénéfices engendrés par l'utilisation des FPGA dans le cadre de la cryptographie. Et, bien que depuis une décennie de nombreux travaux ont abordé ce sujet et qu'il existe de multiples conférences et workshops traitant de l'implantation d'algorithmes cryptographiques sur FPGA, on peut regretter le fait qu'il n'existe pas aujourd'hui de projet visant à mettre au point des FPGA dédiés à la cryptographie. On peut trouver au moins deux raisons à cela. D'une part, les articles traitant des composants cryptographiques reconfigurables sont exclusivement basés sur des architectures à gros grains. A ce titre, on pourra se référer à l'explication donnée par Elbrit et al. dans [ElPa03]. D'autre part, les FPGA sont des produits commerciaux comme les autres et le développement de FPGA dédiés à la cryptographie ou contenant des « *Crypto Slice* » impliquera forcément les acteurs du marché que sont Xilinx, Altera ou Actel. En définitive, c'est certainement le marché qui décidera du développement de cette technologie tout comme c'est lui qui a permis l'ajout de bloc facilitant le traitement du signal dans les FPGA. Il existe toutefois des FPGA dédiés au domaine de la sécurité, embarquant des cœurs de

chiffrement AES des mécanismes de protection, nous pouvons citer l'exemple du circuit ALTERA Cyclone III LS développé pour des applications militaires. Ce composant est accompagné d'une suite logicielle dédiée proposant d'isoler physiquement différentes zones du FPGA, de bloquer la configuration des différentes zones du composant et de chiffrer le fichier de configuration.

2.3.4 **Synthèse : Evaluation des architectures matérielles de crypto-systèmes dans le cadre de la radio logicielle**

Cette dernière partie se propose de faire une synthèse des architectures précédemment présentées avant d'en tirer quelques conclusions sur l'intérêt qu'elles peuvent avoir dans le cadre de la radio logicielle. De façon générale il n'est pas aisé de proposer une comparaison des performances de différentes architectures basées sur des concepts et des technologies différentes et qui ne proposent pas toujours les mêmes fonctionnalités (par exemple des modes de fonctionnement des algorithmes de chiffrement symétrique par blocs). Par ailleurs, Gaj et al. ont montré dans [GKAR10] que de façon générale les résultats d'implantation sur FPGA (i.e. surface silicium consommée, fréquence de fonctionnement, …) dépendaient fortement des outils de synthèse et placement utilisés et des technologies ciblées. Si *ATHENa,* l'outil qu'ils proposent, est capable de comparer différentes implantations d'une même architecture, il ne permet pas de comparer différentes architectures en utilisant les mêmes paramètres d'implantation (i.e. cible technologique, paramètre de synthèse, …). En conclusion, il faut garder à l'esprit que les résultats de synthèse ne donnent pas toujours un aperçu fiable des performances réelles d'une architecture.

Tableau 3: Synthèse des architectures cryptographiques

Architecture	Nom	Année	Algorithmes supportés	Architecture de calcul	Stockage des clés	Crypto. ressources	Compatible PLD	Application
GPP modifié	Custom Xtensa [RRPS02]	2002	DES, 3DES, AES, RSA	Xtensa 32 bits RISC avec ALU modifié	Mémoire principale	Une Crypto-ALU	Non	Sécurité des applications sans fil
	Sbox Instruction [BuMA00]	2000	3DES, IDEA, candidats AES	ALU dédiée aux Sbox	Mémoire principale	Une Sbox	Non	IPSEC, VPN
	Instruction set extension [TiGr06]	2005	AES	Leon 32 bits modifié Sparc V8	Mémoire principale	Opérateur AES	Non	Sécurité des systèmes embarqués
	CryptoBlaze [Xili03]	2003	AES, RSA	Picoblaze 8 bits Xilinx	Mémoire principale	Sbox, multplieur de Galois	Oui	Sécurité des données sur FPGA
Crypto-processeur	Cryptomaniac [WuWA02]	2001	AES, DES, 3DES	Processeur VLIW 4 instructions	Mémoire partagée interne	4 crypto-ALU par processeur	Non	IPSEC, VPN
	Cryptonite [BuHO04]	2004	AES, DES, MD5	2 chemins de données 64 bits	Mémoire principale	2 ALU dédiées	Non	IPSEC, VPN
	CCProc [ThPP08]	2008	Candidats AES	Processeur VLIW	Mémoire principale	4 clusters de Sbox	Non	Chiffrement
Coprocesseur dédié	Processeur AES [HoVe04]	2004	Aes-ECB, CBC-MAC, CCM	Processeur Leon 32 bits	Registre de clé	1 cœur AES	Non	IPSEC, VPN
	Crypto Booster [MTRG99]	1999	IDEA, DES	Cœur matériel reconfigurable	Mémoire de session	2 cœurs AES	Oui	Sécurité des réseaux
	AESTHETIC [CCCC05]	2009	AES-ECB, AES-CBC	Processeur hôte+ cœur AES	Générateur de clés et registre dédié	1 à 3 cœurs AES	Oui	Sécurité des réseaux
	CrCU [CKVS06]	2006	AES-ECB, AES-CBC, SHA	Processeur MOLEN + cœurs AES	Registre de clé	Ressources *CrCU* non limitées	Non	TPM
	AES-MS [PCGV08]	2010	AES-ECB, AES-CBC	Processeur MOLEN + cœurs AES	Registre de clé	2 cœurs AES	Non	VPN
Coprocesseur reconfigurable	Celator [FrPP08]	2008	AES, DES, SHA	4x4 Unité de calcul 8 bits	Mémoire principale	16 PE	Oui	Sécurité des réseaux
	CryptArray [Lomo04]	1999	AES, DES, 3DES	Matrice d'unité de calcul sur 4 bits	Mémoire principale	Nombre variable de PE	Oui	Sécurité des réseaux
	COBRA [ElPa03]	2003	Chiffrement par bloc	4x4 Unité de calcul reconfigurable 8 bits	Mémoire principale	16 PE	Oui	VPN

Tableau 4: Comparaison des débits de différentes architectures

Architecture	Nom [ref]	Cible silicium	Mode de chiffrement	Mbps/MHz	Freq. Max (MHz)	FPGA slice	FPGA RAM	ASIC Portes Surface
GPP modifié	Inst. Set Ext. [TiGr06]	ASIC 130 nm	AES-ECB	0,57	250	-	-	16 K 0,08 mm²
	Intel AES Inst. [Guer10]	ASIC x86 processors	AES-ECB	0.285	> 3 GHz	-	-	-
Coprocesseur dédié	AES Processor [HoVe04]	ASIC 180 nm	AES-ECB	11,6	295	-	-	73 K 0,73 mm²
	CryptoBooster [MTRG99]	FPGA XCV1000	IDEA-ECB	16	33	?	?	-
	AESTHETIC (3 cœurs) [WSHW10]	FPGA XCV2V6000	AES-ECB AES-CBC	36,80	50	27561	0	-
	CrCU [CKVS06]	FPGA XCV2VP30	AES-ECB	0,59	100	847	216Kb	-
	AES-MS2 [PCGV08]	FPGA XCV2VP30	AES-ECB AES-CBC	25,60	100	2161	432 Kb	-
	SANES [GoWB06]	FPGA XCV2VP30	AES-ECB	10,60	37,8	2192	0	-
Crypto-processeur	Cryptomaniac [WuWA02]	ASIC 250 nm	AES-ECB	1,42	360	-	-	1,93 mm²
	Cryptonite [BuHO04]	FPGA XCV2VP30	AES-ECB	5,62	400	1748	32 Kb	-
	CCProc (4 cœurs) [ThSP09]	FPGA XCV4LX200	AES-ECB	6,40	95	18045	?	-
	HCrypt [GFBB10]	FPGA XCV6LX	AES-ECB	1,60	150	7960	9,75 Mb	-
Architecture reconfigurable	Celator [FrPP08]	ASIC 130 nm	AES-CBC SHA 256	0,24 0,19	190	-	-	20 K 0,1 mm²
	CryptArray [Lomo04]	FPGA XCV2VP125	AES-ECB	1,27	81	>50000	0	-
	COBRA [ElPa03]	FPGA XCV1000	AES-ECB	1,40	102	>10000	0	-

Le Tableau 3 donne un aperçu des caractéristiques des architectures cryptographiques précédemment présentées tandis que le Tableau 4 compare le débit de données fourni par différentes architectures (on gardera à l'esprit les remarques faites concernant la pertinence de ce genre de valeurs). On peut d'abord remarquer que les architectures les plus performantes en termes de débit et de latence sont celles qui sont basées sur des coprocesseurs *dédiés*. En ce qui concerne la radio logicielle, le débit n'est la plupart du temps pas un problème contrairement à la latence (cf. Tableau 1). Les architectures dédiées de par leur faible latence sont donc de primes abords intéressantes. Toutefois, leur faible flexibilité n'est certainement pas un avantage lorsqu'elles sont implémentées dans une radio logicielle. Néanmoins, l'architecture multicœur ASTHETIC ou encore l'architecture reconfigurable *CrCU* proposent des solutions intéressantes qui permettent d'améliorer leur flexibilité. Ces

deux architectures peuvent donc avoir un intérêt dans le cadre de la radio logicielle. Elles font donc l'objet d'une étude plus détaillée dans la suite de ce chapitre.

En fait, l'utilisation d'architectures programmables qui privilégient la flexibilité semble une approche plus conforme au principe de la radio logicielle. Néanmoins, cette flexibilité se fait au détriment des performances qui peuvent alors devenir insuffisantes. Afin de pallier à ce problème, nombreuses sont les architectures programmables à proposer un certain degré de parallélisme. *Cryptonite,* avec son architecture VLIW, en est un bon exemple. D'ailleurs, les coprocesseurs reconfigurables tels que *Celator* ou encore *COBRA* sont en quelques sorte un compromis entre architectures dédiées et architectures programmables. Cela leur permet d'avoir des performances élevées tout en restant raisonnablement flexible au détriment toutefois de la surface silicium occupée. Ce compromis fort intéressant dans le cadre de la radio logicielle incite à étudier plus en détail ces deux architectures, c'est ce que nous ferons dans la suite de ce chapitre. Toutefois, on peut regretter le fait que les architectures VLIW ou les architectures reconfigurables présentées soient basées sur un parallélisme à couplage fort (les unités de calculs ne peuvent fonctionner indépendamment les unes des autres). Au contraire, le projet *Cryptomaniac* et son architecture multicœur à base de processeur VLIW semble le seul projet à avoir réellement pris en compte l'aspect multicanal et multistandard des systèmes de communication. Son architecture multicœur faiblement couplée lui permet de traiter chaque paquet de donnée de façon indépendante et autonome et son ordonnanceur lui permet de décharger le processeur principal de l'attribution des tâches à chaque cœur de calcul. Cette architecture fait donc elle aussi l'objet d'une étude détaillée dans la partie suivante.

2.4 Etudes détaillées de quelques architectures

La partie précédente a montré l'intérêt que pourraient avoir les architectures *AESTHETIC*, *CrCU*, *Celator*, *COBRA* et *Cryptomaniac* dans le cadre de la radio logicielle. Dans un premier temps, la structure et le fonctionnement de chacune de ces architectures sont présentés. Puis, dans un second temps, une synthèse des avantages et des défauts de chacune d'elles est réalisée. Cette synthèse servira de point de départ aux travaux présentés plus loin dans ce document.

2.4.1 Version multicœur de l'architecture *AESTHETIC*

L'architecture multicœur *AESTHETIC* [WSHW10] appartient à la famille des architectures à coprocesseur dédié. La version trois cœurs d'*AESTHETIC* fournit un support pour AES en mode ECB et CBC avec des débits allant de 1,29 Gbps à 3,75 Gbps à la fréquence de 102 MHz. Le débit effectif est fonction de la taille des paquets et de la longueur de la clé utilisée. En outre *AESTHETIC* fournit une fonctionnalité pour le moins originale : l'implantation d'AES est **configurable**. En effet, si le standard AES fixe les paramètres des opérateurs qui constituent l'algorithme AES (ex : polynôme générateur pour *SubBytes* et la matrice de *MixColumns*), cela n'empêche pas de les modifier. D'après les concepteurs d'*AESTHETIC*, une telle fonctionnalité permet entre autres de sécuriser le stockage des

données en changeant de polynôme générateur [JiKW04], de changer les paramètres d'AES si ceux-ci venaient à être attaqués avec succès ou encore de développer sa propre version d'AES afin de rendre son système incompatible avec celui des autres dans le but de protéger ses données. Cette dernière affirmation semble **discutable** dans la mesure où la protection par le secret n'est pas conseillée à moins d'avoir mené une étude exhaustive et approfondie de la sécurité des algorithmes cryptographiques mis en œuvre.

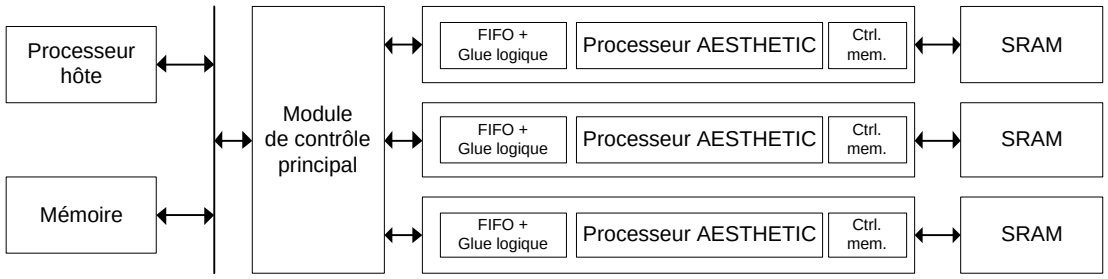


Figure 16: Version multicœur de l'architecture *AESTHETIC* [WSHW10]

L'architecture multicœur d'*AESTHETIC* est illustrée Figure 16. Cette version qui embarque trois cœurs *AESTHETIC* est basée sur la structure suivante : le coprocesseur est connecté à son processeur principal grâce à un bus AHB (*Andvanced High-Performance Bus*) dont le chemin de données est codé sur 32 bits. Le chemin de données du coprocesseur étant codé sur 128 bits, les FIFO d'entrée et de sortie sont conçues de manière à assurer la conversion du format des données entre le bus et le coprocesseur. En plus des cœurs cryptographiques et des interfaces de communication, le coprocesseur embarque un contrôleur principal chargé de configurer les cœurs *AESTHETIC* et de lancer ou stopper les calculs qui s'exécutent dessus. Chacun des cœurs *AESTHETIC* implémente un cœur AES configurable basé sur une architecture séquentielle qui lui permet d'occuper une faible surface de silicium tout en étant compatible avec le mode de chiffrement CBC. A titre d'illustration, voici le déroulement d'une opération de chiffrement :

1. Le processeur hôte accède au registre du contrôleur principal et vérifie qu'au moins un des cœurs est libre afin de le sélectionner le cas échéant. Si aucun cœur n'est libre il attend.
2. Le processeur hôte configure le pointeur vers l'adresse de départ des données à traiter, la clé de chiffrement, le *vecteur d'initialisation* et le registre de contrôle.
3. Le contrôleur principal lance alors l'opération de chiffrement qui consiste d'abord à générer les sous clés AES puis à chiffrer les données.
4. Une fois l'opération de chiffrement effectuée, le contrôleur principal génère une interruption à destination du processeur hôte qui peut alors libérer le cœur de calcul et traiter les données.

De ce détail d'une opération de chiffrement, nous pouvons tirer plusieurs conclusions. D'abord, l'intelligence du contrôleur principal du coprocesseur est plutôt réduite. En effet, c'est le processeur hôte qui est chargé de la gestion des cœurs. Ensuite, aucun mécanisme

n'est fourni dans le but d'incrémenter automatiquement le vecteur d'initialisation alors que c'est une opération couramment utilisée dans le cadre du chiffrement de paquets de données, le vecteur d'initialisation étant incrémenté après le traitement d'un paquet de données. D'un autre côté, le parallélisme à couplage faible de ce coprocesseur (les cœurs de calcul ne communiquent pas entre eux) permet de garantir que les processus de traitement des paquets sont isolés les uns des autres ce qui en termes de sécurité est une bonne chose. Par ailleurs, le chemin critique du coprocesseur est situé à l'intérieur des cœurs de calcul ce qui le rend flexible dans le sens où l'on peut faire varier le nombre de cœurs sans que cela n'ait de conséquences néfastes sur la fréquence de fonctionnement du système.

2.4.2 **Architecture reconfigurable à gros grain COBRA**

L'architecture *COBRA* [ElPa03] propose un mécanisme de reconfiguration original compatible avec une grande variété d'algorithmes cryptographiques symétriques. En effet, cette architecture embarque des unités de calcul et un chemin de données reconfigurables de manière à pouvoir adapter le cheminement du flot de données traitées ainsi que les opérations qui y sont appliquées. Cette construction permet par exemple de configurer le polynôme de réduction d'un multiplieur de Galois ou encore de permuter des bits au sein d'un bloc de données. Contrairement aux FPGA, *COBRA* est basé sur une architecture reconfigurable à gros grain dont le chemin de données est unidirectionnel. Ses concepteurs justifient leurs choix par le fait que les algorithmes cryptographiques opèrent sur un flot de données unidirectionnel et qu'ils utilisent des opérateurs de haut niveau (multiplieurs, additionneurs, …). *COBRA* a été conçu de manière à être compatible avec plus d'une dizaine d'algorithmes de chiffrement symétrique, parmi les plus connus on peut citer *AES*, *Blowfish*, ou encore *Serpent*.

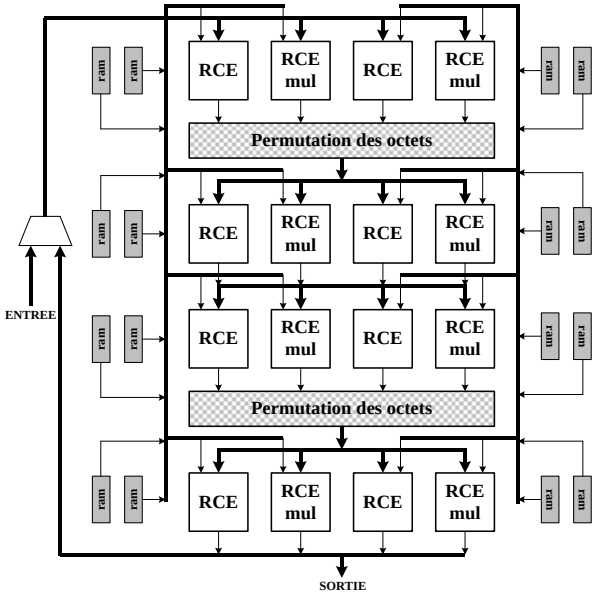


Figure 17: Matrice de calcul de l'architecture *COBRA* [ElPa03]

La Figure 17 illustre la structure générale de l'architecture *COBRA*, on peut y distinguer trois types d'éléments : des *unités cryptographiques reconfigurables* (RCE), des blocs permettant la permutation d'octets et des mémoires. Les RCE sont les pièces maitresses de cette architecture, ce sont elles qui embarquent la quasi-totalité des opérateurs fournis par celle-ci. Ils sont disposés en quatre colonnes embarquant chacune quatre RCE reliées entre elles par des chemins de données sur 128 bits en entrée et 32 bits en sortie le tout formant un chemin de données sur 128 bits. Cette structure est tout à fait adaptée à un algorithme comme AES qui travaille sur des blocs de 128 bits composés de plusieurs mots de 32 bits. On peut aussi remarquer que les RCE sont de deux types différents suivant qu'ils embarquent ou non un multiplieur (RCEmul et RCE). Quel que soit son type, un RCE embarque : des opérateurs logiques, des additionneurs, des tables de substitution, des opérateurs de décalage ou de rotation, des multiplieurs de Galois, des multiplexeurs et un registre. Par ailleurs, bien que la structure des chemins de données soit fixe, les RCE peuvent être désactivées lorsqu'elles ne sont pas utiles à un algorithme. Les mémoires, quant à elles, servent à stocker les valeurs temporaires ou encore les clés de round qui sont utilisées par les RCE. La configuration de la grille de calcul de *COBRA* se fait par l'intermédiaire d'un programme stocké dans une mémoire spécifique et écrit à l'aide d'un jeu d'instructions VLIW codé sur 80 bits. Les instructions servent à configurer les RCE, les activer ou les désactiver, configurer les blocs RAM ou encore gérer les interfaces d'E/S. Avec cette version de *COBRA*, il est possible de configurer sur la matrice deux round d'AES ou de RC6 ou encore un round de Serpent. Cela permet d'atteindre des débits allant de 25 Mbps avec Serpent à 593 Mbps avec AES pour une fréquence de 60 MHz et un total de portes équivalentes ASIC de 6,7 millions de portes.

L'architecture *COBRA* surprend par sa flexibilité, car elle fournit un support pour l'accélération matérielle de plus d'une dizaine d'algorithmes cryptographiques. Cette généricité est particulièrement avantageuse dans le cadre de la radio logicielle. Toutefois, l'implantation de ce type d'architecture nécessite une grande surface silicium. Par ailleurs, la capacité qu'ont certains FPGA SRAM à être reconfigurés partiellement tend à réduire l'intérêt d'une telle approche. En effet, il est plus simple de reconfigurer partiellement un FPGA avec différents accélérateurs dédiés en fonction des besoins que de concevoir un accélérateur multimode. Il faut malgré tout relativiser cette remarque dans le sens où la vitesse de reconfiguration partielle des FPGA actuels n'est pas encore suffisante pour envisager une reconfiguration « temps réel » des circuits.

2.4.3 **Architecture reconfigurable à gros grain Celator**

Tout comme *COBRA, Celator* est un coprocesseur cryptographique reconfigurable à gros grain [FrPP08]. Il est basé sur une architecture systolique qui s'articule autour d'une matrice configurable (cf. Figure 18) de 4x4 unités de calcul (PE). La matrice à deux dimensions de *Celator* est particulièrement bien adaptée à l'exécution d'algorithmes de chiffrement symétrique, car, en général, ceux-ci travaillent sur des données rangées dans un tableau à deux dimensions (ex: matrice d'état d'AES). *Celator* est présenté comme un compromis entre une architecture à base de GPP et un accélérateur matériel dédié. La plateforme

complète de *Celator* s'articule autour d'un processeur ARM, d'une mémoire, d'une file de type FIFO et d'un coprocesseur *Celator*. La FIFO permet au processeur et à *Celator* de fonctionner à des fréquences différentes. *Celator* est composé en lui-même d'une grille de PEs, d'un contrôleur et d'une RAM locale appelée CRAM qui sert de support de stockage aux données en cours de traitement et au contenu des S-Box. L'architecture *Celator* est capable d'exécuter les algorithmes AES, DES et SHA-256.

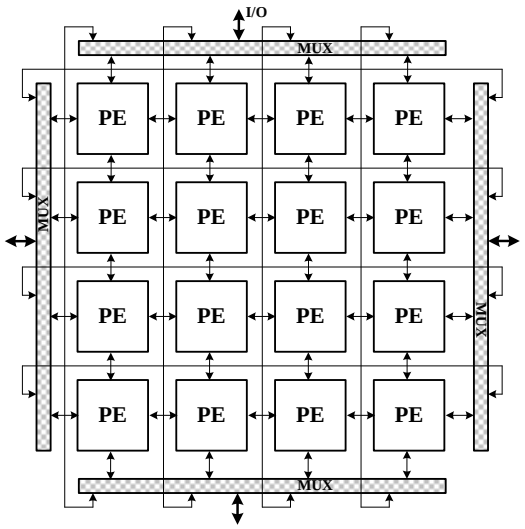


Figure 18: Matrice de calcul de l'architecture *Celator* [FrPP08]

L'architecture des PEs est gardée confidentielle par Atmel toutefois, chaque unité de calcul contient:

- 1 ALU qui embarque :
  - Un opérateur *Xtime* (multiplieur de Galois pour AES).
  - Des opérateurs logiques (XOR, AND, NOT).
  - Un additionneur
  - Un opérateur de décalage fixe d'un bit.
- 2 registres 8-bits (A, B)
- 4 ports d'E/S sur 8-bits
- 2 multiplexeurs qui permettent de sélectionner l'entrée et la sortie d'un PE parmi les ports N/E/W/S.
- 2 multiplexeurs permettant de sélectionner l'entrée des registres A et B.

La structure de *Celator* est plus fine que celle de *COBRA* dans le sens ou les PE sont reliés entre eux par des chemins de données codés sur 8 bits. Il s'ensuit que *Celator* ne fournit qu'un débit de 47 Mbps à une fréquence de fonctionnement de 190 MHz lorsqu'il exécute l'algorithme AES. Par ailleurs, aucun mécanisme de génération de sous-clés n'est

implémenté dans *Celator*. Les clés de round doivent donc être calculées au préalable par un autre composant. Pour finir, la structure de *Celator* semble moins efficace que celle de *COBRA*. En effet, l'architecture de *Celator* semble peu adaptée au traitement d'un flot de données car il est par exemple impossible de dérouler plusieurs rounds d'un algorithme ou même d'exécuter un round par cycle tout simplement.

2.4.4 **Coprocesseur reconfigurable dédié *CrCU***

Dans [CKVS06], Chaves et al. présentent une architecture de TPM (*Trusted Platform Module*) embarquant un ou plusieurs coprocesseurs reconfigurables appelés *CrCU* (*Cryptographical Computation Unit*) connectés à un processeur hôte par l'intermédiaire d'un bus de communication. L'architecture *CrCU* est exclusivement destinée à être implémentée sur FPGA. De cette manière, l'accélérateur cryptographique embarqué dans chaque coprocesseur *CrCU* est reconfigurable statiquement (à la mise sous tension) ou dynamiquement (en cours de fonctionnement). La Figure 19 illustre l'architecture du TPM. On y retrouve : un processeur généraliste, une mémoire interne permettant le stockage des données critiques (certificats cryptographiques, signature numérique maître), un contrôleur de bus, un périphérique d'E/S ainsi que plusieurs coprocesseurs *CrCU*. Le processeur hôte est chargé des tâches de haut niveau (génération et gestion des clés, couches haut niveau des protocoles cryptographiques) tandis que les coprocesseurs *CrCU* sont chargés de fournir un support matériel permettant l'accélération des algorithmes cryptographiques. Ces unités peuvent être reconfigurées en fonction des besoins du système : une application peut par exemple utiliser un protocole basé sur 3DES/RSA/SHA tandis qu'une autre peut utiliser AES/RSA/Whirlpool. Le processus de reconfiguration implique une modification des composants logiciels et matériels. Or, cette modification doit se faire dans des conditions de sécurité satisfaisante et il s'ensuit que l'authenticité d'un composant doit être vérifiée avant qu'il ne soit chargé. Le TPM embarque à cet effet un coprocesseur *CrCU* (en gris sur la Figure 19) dédié à cette tâche et configuré statiquement.

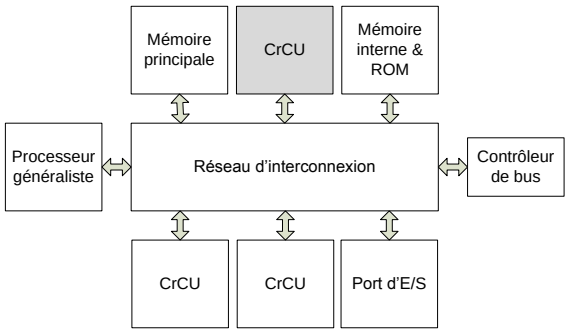


Figure 19: Architecture du TPM [CKVS06]

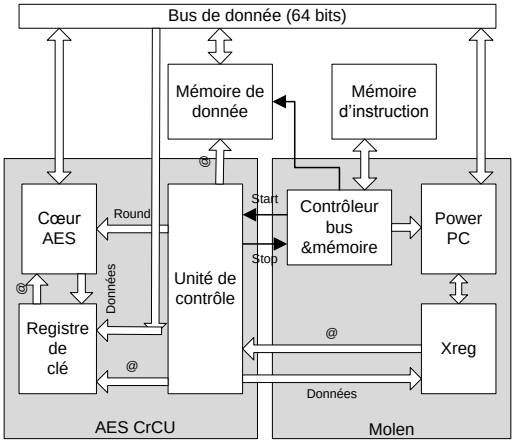


Figure 20: Architecture AES-*CrCU* [CKVS06]

Afin de montrer la faisabilité de leur architecture, Chaves et al. ont développé un prototype de TPM embarquant un coprocesseur *CrCU*. Ce dernier est contrôlé grâce à un processeur polymorphique *MOLEN* [VWGB04] qui fournit les couches d'abstractions nécessaires au contrôle d'un coprocesseur reconfigurable. La Figure 20 illustre l'architecture d'un cœur *CrCU* configuré pour exécuter l'algorithme AES. On y retrouve un processeur *MOLEN* agissant comme hôte ainsi qu'un accélérateur AES séquentiel et son unité de contrôle. La logique de contrôle présente autour du processeur Power PC embarqué dans le processeur *MOLEN* permet quant à elle la bonne communication entre l'accélérateur AES et le processeur Power PC. Le prototype développé n'embarquant pas de mécanisme permettant la reconfiguration partielle des cœurs *CrCU*, plusieurs versions de ces cœurs ont été développées. Les algorithmes suivant sont supportés : AES-ECB/CBC, DES, SHA-128/256 et Whirlpool.

2.4.5 **Cryptomaniac**

L'architecture *Cryptomaniac* [WuWA02] est une architecture de cryptoprocesseur mono ou multicœur spécialement conçue pour les applications cryptographiques sur ASIC. Cryptomaniac repose sur une architecture MIMD (*Multiple Instruction on Multiple Data*) qui exécute des instructions VLIW de 128 bits de large. Chaque instruction VLIW est composée de quatre instructions 32 bits qui s'exécutent de façon concurrente. Une instruction 32 bits peut traiter jusqu'à trois opérandes en lecture et un en écriture. Le séquençage des instructions (dépendance entre variables, etc.) est fait de manière statique à la main ou grâce à un compilateur. Lorsque qu'un cœur *Cryptomaniac* exécute quatre instructions combinées en parallèle et qu'il fonctionne à sa fréquence maximale de 360 Mhz un débit maximum d'environ 65Mo/s est atteint pour l'algorithme AES.

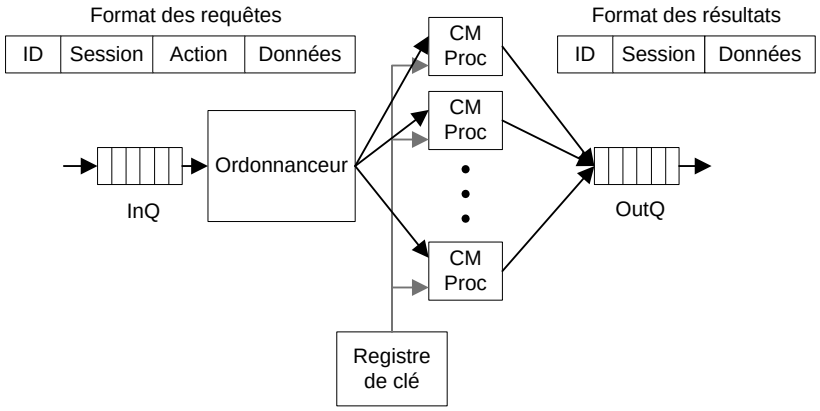


Figure 21: Architecture *Cryptomaniac* [WuWA02]

La Figure 21 présente l'architecture. Le processeur hôte est relié à *Cryptomaniac* via une file d'entrée *InQ* et une file de sortie *OutQ*. Un ordonnanceur se charge par la suite de distribuer les requêtes aux différents cœurs de processeur qui fonctionnent en parallèle. Chaque requête entrante est taguée avec les informations suivantes :

- *Un identifiant de requête* : Pour que l'hôte puisse établir une correspondance entre requêtes entrantes et sortantes.
- *Un identifiant de session* : Pour spécifier quelle session utiliser (clé, algorithme cryptographique).
- *Un code de requête* : Pour indiquer l'action à exécuter.

Il existe trois types de requêtes permettant d'ouvrir une session, fermer une session et chiffrer/déchiffrer un message. Lors d'une ouverture de session, l'algorithme, le mode de chiffrement et la clé doivent être spécifiés, les sous-clés sont ensuite générées. Par la suite les informations de session sont stockées dans une mémoire appelée *Keystore*.

Dans le cas ou plusieurs cœurs de processeur sont effectivement utilisés, l'ordonnancement des requêtes se fait de la façon suivante. L'*ordonnanceur de requête* récupère une requête dans la file *InQ*, puis deux cas se présentent :

- La session est déjà ouverte sur un processeur et celui-ci est disponible. Dans ce cas la requête lui est expédiée.
- Aucun processeur avec la bonne session ouverte n'est disponible. Dans ce cas la requête est expédiée au processeur le moins récemment utilisé, car c'est celui qui a le plus de chance d'avoir un emplacement de session libre.

La Figure 22 détaille l'architecture d'un cœur de calcul appelé *Processing Element* (PE). Les PEs peuvent exécuter quatre instructions de façon concurrente grâce aux *unités fonctionnelles* (cf. Figure 23) qu'ils embarquent. Les PEs sont composés de quatre étages de pipeline dont :

- L'étage de prédiction de branchement et d'adressage des instructions.
- L'étage de décodage des instructions et de lecture des données dans les registres processeurs.
- L'étage d'exécution et des accès mémoire.
- L'étage de *Write Back*.

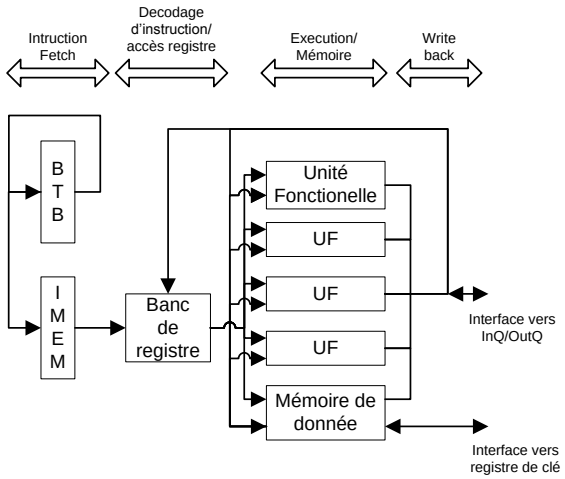


Figure 22: Architecture d'un cœur *Cryptomaniac*

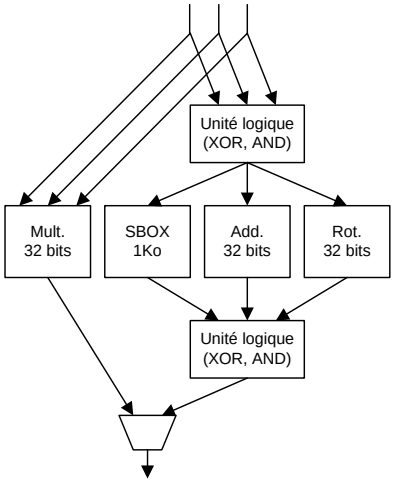


Figure 23: Architecture d'une unité de fonctionnelle

Les instructions sont exécutées de la façon suivante : d'abord une instruction VLIW est chargée depuis la mémoire programme, ensuite les quatre sous instructions sont décodées et accèdent en même temps au banc de registres. A l'étage d'exécution jusqu'à quatre opérations sont effectuées en parallèle. Pour finir les résultats des calculs sont enregistrés dans le banc de registres ou sauvegardés dans le *Keystore*.

L'étage de prédiction de branchement s'articule autour d'un *Branch Target Buffer* (BTB) [Perl93]. Le BTB contient les adresses cibles des branchements et considère que tous les branchements sont pris, ce qui est souvent le cas pour les algorithmes cryptographiques. D'après [Perl93] seize entrées sont suffisantes pour le BTB et une mémoire de 1Ko est suffisante pour contenir un programme implantant l'un des algorithmes cryptographiques compatibles avec *Cryptomaniac*.

Les instructions accèdent au banc de registres durant leur décodage, le banc de registres nécessite donc trois accès en lecture, un en écriture pour chacune des quatre instructions (soit douze entrées et quatre sorties). Les opérations et instructions du type *load/store* sont exécutées à l'étage EX/MEM, les mémoires de stockage des données n'ont pas besoin d'avoir une taille importante étant donné que les S-BOX sont directement implantées au niveau des unités fonctionnelles. D'après les auteurs, 4Ko de mémoire sont suffisant. L'étage d'exécution inclut donc quatre S-BOX de 1Ko dans chaque unité fonctionnelle.

Chacune des S-BOX contient des tables de substitutions et chaque table est alignée sur une page mémoire afin de faciliter l'adressage de celle-ci.

2.4.6 **Synthèse**

Cette partie propose une synthèse de l'étude bibliographique précédente en analysant chaque architecture au travers du prisme de la radio logicielle. L'adéquation de chacune d'elles vis-à-vis des contraintes *d'interopérabilité*, *d'évolutivité*, de *performance*, de *parallélisme* (radio multicanal) et de *sécurité* est étudiée.

Interopérabilité : Concernant l'interopérabilité, on peut distinguer deux catégories d'architecture : les architectures dédiées d'un côté et les architectures programmables et/ou reconfigurables de l'autre. Dans la première catégorie, on retrouve notamment *AESTHETIC* et dans une moindre mesure *CrCU*. Pour sa part, *AESTHETIC* propose une implantation paramétrable d'AES dont l'interopérabilité est réduite dans la mesure où elle n'est compatible qu'avec des variations de l'algorithme AES. En ce qui concerne *CrCU*, l'approche est différente. *CrCU* est présentée comme une architecture tirant profit de la reconfigurabilité des FPGA. Malheureusement, aucune technique de reconfiguration *partielle* n'est présentée dans ces travaux ce qui fait de *CrCU* une architecture uniquement reconfigurable de façon *statique*. Or, la reconfiguration *complète* d'un FPGA embarquant plusieurs cœurs *CrCU* n'est pas envisageable, car ceux-ci fonctionnent indépendamment les uns des autres. Il s'ensuit que le niveau d'interopérabilité de *CrCU* est limitée aux algorithmes implémentés et chargés lors de l'initialisation du système en l'occurrence : AES, DES, SHA et Whirlpool. Toutefois, l'ajout d'un mécanisme de reconfiguration partielle à CrCU améliorerait grandement sa flexibilité et par là même son interopérabilité.

Dans l'autre catégorie, on retrouve *Celator*, *COBRA* et *Cryptomaniac*. L'interopérabilité de *Celator* et *COBRA* est plutôt bonne dans le sens ou passer d'un algorithme à un autre est aisée grâce au caractère reconfigurable de leur chemin de données et de leurs opérateurs de calcul. On pourra toutefois regretter que la structure de *Celator* soit trop proche de celle d'AES alors que cela n'a pas d'impact positif significatif sur les performances. Par exemple, la structure de *Celator* ne permet pas le support de l'algorithme RC4 qui est encore largement utilisé malgré la découverte de plusieurs faiblesses [TeBe09]. L'architecture *COBRA* a été conçue de manière à permettre le support de plusieurs des algorithmes de chiffrement symétriques (une dizaine environ). Son chemin de données et ses opérateurs sont donc probablement suffisamment génériques pour qu'elle soit compatible avec de futurs algorithmes. Pour finir, l'architecture *Cryptomaniac* suit le schéma de construction des processeurs généralistes VLIW en proposant en outre des opérateurs spécifiques à la cryptographie et notamment des tables de substitution. Cela rend *Cryptomaniac* compatible avec la majorité des algorithmes de chiffrement symétriques si l'on accepte que cette flexibilité se fasse au détriment des performances.

Evolutivité : L'évolutivité d'une architecture peut être facilement quantifiée. En réalité, celle-ci dépend plus de la cible technologique utilisée pour l'implantation d'une architecture que de l'architecture elle-même. Que ce soit *AESTHETIC*, *Celator*, *COBRA* ou encore

Cryptomaniac aucune de ces architectures n'est à proprement parler évolutive, car elles sont basées sur une technologie ASIC. On pourra toutefois remarquer que la généricité de *CORBA* et de *Cryptomaniac* fait que ces architectures pourraient être compatibles avec les prochaines générations d'algorithmes de chiffrement. Par ailleurs, il est toujours possible de porter sur FPGA ces architectures si l'on y apporte certaines modifications architecturales. Au contraire, *CrCU* est spécifiquement développé dans le but d'être implanté dans des composants reconfigurables tels que les FPGA. La seule limite à l'évolutivité d'une architecture telle que *CrCU* est définie par les performances, caractéristiques et fonctionnalités du composant reconfigurable dans laquelle elle est implémentée. Cette limite n'est en général pas spécialement contraignante, surtout lorsque l'on considère les améliorations régulièrement apportées aux composants reconfigurables. Néanmoins, un problème se pose en ce qui concerne l'évolution des moyens de sécurisation des composants cryptographiques. Que ceux-ci soient basés sur une technologie ASIC ou FPGA, il est difficile, voire impossible, de faire évoluer les contremesures qui y sont embarquées. Cette constatation est notamment à l'origine de travaux sur des contremesures de haut niveau agissant au niveau algorithmique ou architectural [KaBG08].

Performance : Le Tableau 4 montre que les architectures les plus performantes en termes de débit et latence sont les architectures dédiées. De manière générale, plus la granularité des opérateurs matériels mis à contribution dans un algorithme est grosse plus les performances sont importantes. Les architectures dédiées ayant la granularité la plus grosse, elles sont donc les plus performantes (pour une occupation silicium et une technologie donnée). On ne s'étonnera donc pas du fait qu'*AESTHETIC* ou *CrCU* proposent les performances les plus élevées. Toutefois, ce haut niveau de performance est atteint au détriment de la flexibilité et de l'interopérabilité. En ce sens, la proposition faite par les concepteurs de *CrCU* d'utiliser la reconfiguration statique et/ou dynamique des FPGA afin de remplacer un accélérateur matériel par un autre permet de profiter à la fois des performances des architectures dédiées tout en conservant en grande partie la flexibilité des architectures programmables. Reste qu'il n'est pas toujours possible d'utiliser la reconfiguration partielle. Soit parce qu'une technologie ASIC est utilisée, soit parce que la vitesse de reconfiguration du FPGA n'est pas suffisante pour répondre aux exigences du cahier des charges du système à implanter. Dans ce cas, le concepteur doit se tourner vers une architecture reconfigurable similaire à celle de *COBRA*, l'interopérabilité de *Celator* pouvant être jugée insuffisante pour une radio logicielle. La Figure 11 représentant l'espace de conception des architectures cryptographiques illustre particulièrement bien ces observations. Pour finir, on pourra remarquer que même si les performances de l'architecture *Cryptomaniac* sont limitées par son caractère programmable, il est possible d'en améliorer les performances en augmentant le nombre de *processing elements* qu'elle embarque.

Parallélisme (contexte de la radio multicanal) : Dans le contexte de la radio logicielle multicanal, les utilisateurs doivent pouvoir ouvrir plusieurs canaux de communication simultanément. Chaque canal de communication utilise des clés de chiffrement différentes et éventuellement un protocole cryptographique différent. On en déduit que la capacité de

changer rapidement de clé ou d'algorithme de chiffrement est une caractéristique fondamentale des composants cryptographiques multicanaux et multistandards. Commençons d'abord par dire que *Celator* n'intègre pas de mécanisme de génération des clés de round, il est donc difficile d'évaluer sa capacité à changer rapidement de clé de chiffrement. En ce qui concerne *CrCU*, *COBRA*, ou encore *AESTHETIC* des mécanismes de génération et de stockage des sous clés sont prévus. Les sous clés n'ayant pas besoin d'être générées par un processeur généraliste puis envoyées au coprocesseur cryptographique (seule la clé secrète est envoyée), la latence du système en est grandement réduite. De plus, les architectures *AESTHETIC* et *CrCU* étant multicœur, il n'est alors plus forcément nécessaire d'attendre la fin d'une tâche pour en lancer une autre. Néanmoins, la distribution des tâches sur chacun des cœurs est gérée par le processeur principal ce qui le surcharge inutilement. L'architecture *Cryptomaniac* propose de solutionner ce problème en intégrant un ordonnanceur de tâche directement au sein de l'architecture du cryptoprocesseur. Celui-ci permet par exemple d'attribuer de façon transparente pour le processeur hôte une session à un cœur de calcul. De cette façon, les temps nécessaires aux changements de contexte peuvent être économisés. Pour finir, les cœurs fonctionnent indépendamment des autres, on ne rencontre donc pas les problèmes d'inter-blocage qui pourraient apparaitre entre différentes sessions partageant une même ressource dans une architecture implémentant un parallélisme à couplage fort. A titre d'exemple, si l'on utilise déjà 64 bits du chemin de données de *COBRA* pour une session utilisant l'algorithme DES, il n'est pas possible de tirer profit des 64 bits restants.

Sécurité : La sécurité est une problématique complexe de la conception des composants cryptographiques. Or, nous avons vu qu'elle n'était pas systématiquement prise en compte lors de l'élaboration de nouvelles architectures destinées aux applications cryptographiques. Si l'on met de côté les contremesures matérielles pour s'intéresser exclusivement au niveau de sécurité inhérent aux architectures, on constate que parmi les cinq architectures présentées seule *Cryptomaniac* permet une isolation efficace des clés. Que ce soit *AESTHETIC*, *COBRA* ou encore *Celator* aucune d'elles ne propose de mécanisme de stockage des clés cryptographiques. L'architecture *CrCU* propose quant à elle une mémoire de stockage dédiée aux clés de chiffrement. Malheureusement, cette mémoire est directement connectée au bus de communication partagé par l'ensemble du système, on ne peut donc pas exclure une fuite des clés de chiffrement au travers des interfaces d'E/S du système. *Cryptomaniac* propose elle aussi une mémoire dédiée au stockage des clés. Néanmoins, les travaux présentés n'indiquent pas comment les clés sont envoyées vers cette mémoire. De plus, les clés doivent être stockées dans le banc de registre d'un cœur avant qu'elles ne puissent être utilisées, rien n'empêche donc leur fuite par les interfaces d'E/S du cœur. Toutefois, en proposant un ordonnanceur spécifique, l'architecture *Cryptomaniac* améliore le niveau d'isolation entre les différents canaux de communication. Inversément, le fait d'attribuer les tâches de gestion des canaux et d'ordonnancement des paquets au processeur hôte est contraire au principe d'isolation fonctionnelle des composants. A terme, cela peut faciliter la mise en œuvre d'attaques logicielles et/ou matérielles visant à compromettre la sécurité du système.

Tableau 5: Adéquation des architectures de crypto-système présentées avec les besoins de la radio logicielle multicanal sécurisée

Architecture	Adéquation avec le besoins de la radio logicielle sécurisée				
	Interopérabilité	**Evolutif**	**Performances**	**Parallélisme**	**Sécurité**
AESTHETIC	Faible	Non	Elevées	Moyen	Faible
COBRA	Elevée	Non	Moyennes	Faible	Faible
Celator	Moyenne	Non	Faibles	Faible	Faible
CrCU	Faible (Elevée si RP[13])	Oui	Elevées	Moyen	Faible
Cryptomaniac	Elevée	Non	Moyennes	Elevée	Faible ou Moyenne si chemin de données séparé pour les clés

2.5 **Conclusion**

Le Tableau 5 récapitule les observations de la partie précédente. Les cellules grisées du tableau permettent de souligner les points forts de chacune des architectures présentées. La répartition de ces cellules montre qu'aucune des architectures ne remplit l'ensemble des besoins de la radio logicielle sécurisée. D'ailleurs, aucune de ces architectures ne fournit un niveau de sécurité suffisant pour les radios logicielles sécurisées à destination du marché militaire. En effet, les clés sont stockées dans les registres généraux des cryptoprocesseurs et autres accélérateurs matériels et aucun mécanisme de protection n'est présenté. Or, le supplément sécurité à la SCA stipule clairement qu'aucune clé de chiffrement rouge (au contraire des clés de chiffrement noires) ne doit pouvoir sortir du système de chiffrement. La mise au point d'une ressource matérielle permettant la prise en charge et l'accélération des services cryptographiques passe donc par l'utilisation :

- d'architectures hybrides basées sur l'utilisation conjointe de différents types d'architecture (ex : architecture reconfigurable multicœur).
- d'architectures assurant une isolation stricte entre les parties rouge et noire de la radio pour empêcher notamment la fuite des clés cryptographiques par les interfaces d'E/S du système.

La conception d'une telle ressource matérielle n'est pas triviale car comme nous l'avons déjà dit : certains des besoins de la radio logicielle sont antagonistes. Le concepteur se trouve donc contraint à faire des compromis lors des différentes étapes de la mise au point d'une architecture adaptée à la radio logicielle. Afin d'aider le concepteur dans son travail, la suite de ce document se propose d'apporter quelques éléments de réponse à cette problématique en proposant notamment une architecture de cryptoprocesseur multicœur.

[13] Reconfiguration Partielle

Chapitre 3

Conception d'un crypto-système pour la radio logicielle sécurisée

3.1 Introduction

Le chapitre précédent a montré la nécessité de mettre au point un sous-système cryptographique dédié à la radio logicielle sécurisée. Ce sous-système a pour objectif de proposer une solution répondant à travers un compromis à l'ensemble des besoins de ce type de radio que sont : l'*interopérabilité*, l'*évolutivité*, les *performances*, l'*aspect multi-utilisateurs* et la *sécurité*. Si les sous-systèmes cryptographiques qui existent (voir chapitre précédent) ne proposent pas un tel compromis, cela est en partie dû aux limitations des technologies actuelles. Cependant, l'évolution rapide des circuits reconfigurables nous permet de proposer une nouvelle solution. Effectivement, en utilisant ces circuits il est aujourd'hui envisageable de concevoir des architectures à la fois performantes et flexibles permettant l'implantation de composants cryptographiques performants, interopérables et évolutifs.

Dans la suite de ce chapitre, nous présenterons le concept du MCCP (*Multi Core Crypto Processor*) qui repose sur l'utilisation de plusieurs cœurs cryptographiques à la fois programmables et reconfigurables. Ces caractéristiques font du MCCP un cryptoprocesseur flexible et interopérable répondant aux besoins de la radio logicielle sécurisée. Sa conception repose sur l'utilisation des dernières technologies de circuit FPGA mais aussi sur la mise en œuvre de méthodologies de conception destinées à en améliorer le niveau de performance, de fiabilité et de sécurité. Ce chapitre n'a pas pour objectif de fournir un plan d'implantation détaillé du MCCP. Il vise plutôt à transmettre au lecteur les éléments de la méthodologie de conception mise en œuvre dans le développement d'un sous-système cryptographique de type MCCP. L'implantation concrète et détaillée d'un prototype du MCCP est présentée au chapitre suivant.

Ce chapitre est divisé en cinq parties chacune d'elles traitant un aspect différent de la conception d'un MCCP. Avant d'entamer la présentation du MCCP, la partie 3.2 présente deux fonctionnalités mises en œuvre lors de la conception et l'utilisation du MCCP, qui sont spécifiques aux composants FPGA: la *reconfiguration partielle dynamique* et *l'isolation électrique* de différents circuits configurés dans un FPGA. La partie 3 présente l'architecture générique d'un MCCP en décrivant sa structure générale ainsi que la manière de l'intégrer et de l'interfacer dans son environnement de fonctionnement qui peut être un *Crypto Sub System*. Par la suite, les parties 3.4 et 3.5 présentent plus en détail le fonctionnement de chacun des éléments composant cette architecture. Pour finir, la partie 3.6 traite de

l'ordonnancement des paquets et de l'attribution de leur traitement aux différentes ressources cryptographiques disponibles au sein d'un MCCP.

3.2 Etudes préliminaires : reconfiguration partielle et architecture en coupure sur FPGA

Avant d'entamer la description de l'architecture MCCP, il est utile de donner d'avantage de précisions sur les techniques et méthodologies de conception propres à l'utilisation des composants FPGA. Pour cela, nous nous appuierons sur des travaux passés et sur la documentation fournie par les fabricants de FPGA. Dans un premier temps, la problématique de la reconfiguration partielle dynamique et sécurisée sera abordée. Puis, dans un second temps, nous aborderons le problème de l'isolation « physique » de différentes parties d'un circuit implanté sur FPGA dans le but de répondre aux besoins du *principe de partitionnement rouge/noir*.

3.2.1 Reconfiguration partielle, dynamique et sécurisée du matériel

Le principe de *reconfiguration dynamique partielle*, introduit par Xilinx pour sa famille de FPGA Virtex puis par Altera pour sa famille de FPGA Stratix V, a permis d'agrandir l'espace de conception d'architectures matérielles. Grâce à la *reconfiguration dynamique partielle*, il est possible de reconfigurer seulement une partie du circuit implanté sur le FPGA tout en gardant le reste du circuit fonctionnel. On peut alors envisager la conception d'applications basées sur la reconfiguration à la volée d'une partie du FPGA à la manière du changement de processus actif d'un processeur généraliste.

Avec les outils Xilinx, le développement d'une application tirant profit de la *reconfiguration dynamique partielle* se fait en trois étapes [Xili11a]. D'abord, le concepteur doit décrire en langage HDL le circuit électronique en séparant la description des parties du circuit qui seront par la suite partiellement reconfigurables. Puis, lors des étapes de placement et de routage, le concepteur définit un certain nombre de zones qui seront reconfigurables et leur attribue plusieurs descriptions VHDL, une seule pouvant être active à la fois. Pour chaque description VHDL définie et chaque zone reconfigurable, un fichier de configuration partielle ou *bitstream partiel* est généré. En outre, un dernier fichier de configuration est généré pour permettre la configuration du FPGA dans son état par défaut. Enfin, lors des phases de fonctionnement, un processeur se charge de reconfigurer le FPGA grâce aux fichiers de configuration partielle générés lors des étapes de placement et de routage. Le processus de reconfiguration dynamique se fait par le biais d'un port externe ou par un circuit interne, appelé *Internal Configuration Access Port* (ICAP) chez Xilinx, présent dans l'ensemble des circuits de la famille Virtex à partir du circuit Virtex II [Xili10a]. Cette interface permet au FPGA de s'*auto-reconfigurer* de manière totalement autonome.

Une telle technologie rend par exemple possible le passage à la volée d'un protocole de communication à un autre (ex. : WiFi vers Bluetooth) tout en n'ayant à chaque instant qu'une seule chaîne matérielle de modulation/démodulation configurée sur le FPGA. La

réalisation d'une telle application passe par la conception d'unités de contrôle matérielles ou logicielles pour la reconfiguration partielle. Dans [BeMS08], Beretta et al. proposent une architecture de SoC (cf. Figure 24) utilisant la technique de reconfiguration partielle disponible sur les circuits Xilinx Virtex II Pro. Cette architecture s'articule autour d'un processeur *Microblaze* qui contrôle une interface *ICAP* au travers d'un noyau Linux. Ceci permet entre autres de modifier les fonctionnalités d'un ou plusieurs coprocesseurs reconfigurables de manière à fournir un support matériel adapté aux processus s'exécutant sur le processeur hôte.

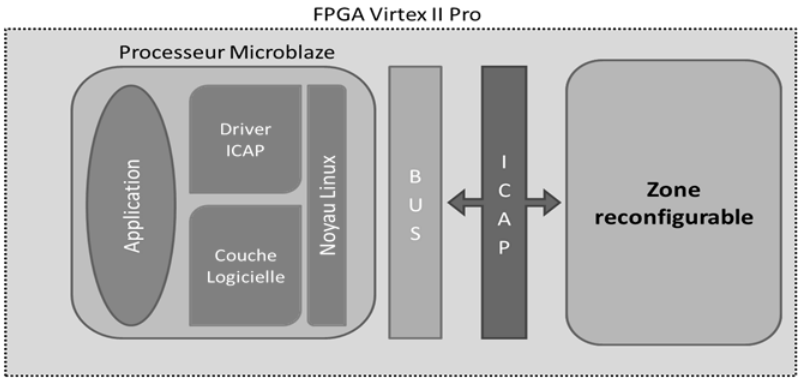


Figure 24 : Architecture d'un SoC utilisant un noyau Linux pour contrôler le processus de reconfiguration partielle [BeMS08]

En ce qui concerne la *reconfiguration partielle dynamique*, il faut retenir que la reconfiguration d'une partie d'un FPGA est plus rapide que la reconfiguration totale d'un FPGA et que ce processus n'a pas d'influence sur le fonctionnement du reste du FPGA. Cependant, il est nécessaire de prendre en compte un certain nombre de considérations techniques lorsque l'on souhaite utiliser la reconfiguration partielle. Tout d'abord, la rapidité du processus de reconfiguration conditionne directement la façon dont cette technologie peut être utilisée. En effet, si le processus de reconfiguration est suffisamment rapide et qu'il n'introduit pas de délais gênants, il est possible d'envisager la reconfiguration en « temps réel » des cœurs cryptographiques. Malheureusement, les délais de reconfiguration partielle (de l'ordre de la centaine de millisecondes [LKLJ09a] dans certains cas) sont du même ordre de grandeur que la latence maximale admissible par certaines applications (par exemple la VoIP ou la vidéoconférence) ce qui en limite l'intérêt dans le domaine des communications « temps réel ». L'optimisation de ces délais peut se faire de plusieurs manières : en dimensionnant efficacement les zones reconfigurables, en mettant en œuvre des mécanismes de décompression rapide des bitstreams, en stockant les bitstreams en cache ou en améliorant les performances des IP matérielles dédiées à la reconfiguration. L'amélioration des IP matérielles est laissée aux constructeurs de FPGA, bien qu'il soit toujours possible pour l'utilisateur de sur-cadencer ces systèmes. Quant au dimensionnement des zones reconfigurables, le concepteur bénéficie d'une marge de manœuvre réduite. Reste donc la mise en cache des bitstreams [CZSB08, LKLJ09b] et leur compression [HUWB04]. La mise

en cache des bitstreams partiels à l'intérieur du FPGA permet d'une part de réduire les temps d'accès au bitstream et d'autre part permet d'éviter de stocker dans une mémoire externe les bitstreams sous forme chiffrée. Toutefois, la quantité de mémoire disponible dans un FPGA est relativement limitée. La compression des bitstreams joue alors un rôle important lorsqu'il s'agit de les mettre en mémoire cache. Par ailleurs, la compression des bitstreams permet aussi de réduire les temps d'accès totaux aux mémoires qui servent au stockage permanent des bitstreams.

La mise en œuvre sécurisée de ces techniques pose le problème de la confidentialité, de l'intégrité et de l'authenticité des *bitstreams partiels* chargés sur le FPGA. Certains FPGA supportent le chiffrement des *bitstreams statiques* avec AES (Xilinx, Altera et Actel) ainsi que leur authentification (Virtex 6 de Xilinx). Toutefois, aucun FPGA n'intègre de mécanisme permettant d'assurer la *confidentialité* ou l'*authenticité* d'un *bitstream partiel*. L'implantation de ces mécanismes est laissée à la charge de l'utilisateur [CHLM06, KMKS08, Zein05]. Le niveau de sécurité d'une chaîne de reconfiguration partielle dépend donc de l'implantation qui en est faite par son concepteur. En outre, l'ICAP à la capacité d'accéder en lecture au *bitstream* du FPGA, cela a pour effet de rendre inefficace le chiffrement du *bitstream* si l'accès en lecture à l'ICAP n'est pas protégé. De plus, rien n'empêche un attaquant de reconfigurer dynamiquement une zone configurée statiquement. Plus précisément, un *bitstream* partiel embarque des instructions qui spécifient la ou les zones qui doivent être reconfigurées. Un ICAP sécurisé devrait vérifier que ces zones correspondent à des zones effectivement reconfigurables sans quoi l'intégrité du *bitstream* statique déjà chargé sur le FPGA pourrait être menacée.

Un dernier problème vient du fait que les technologies SRAM utilisées pour la fabrication des FPGA sont gravées de plus en plus finement ce qui implique une plus grande sensibilité aux perturbations extérieures (ex. : perturbations électromagnétiques). Afin de protéger le système contre les fautes de type *Single Event Upset* naturelles ou intentionnelles (attaques par injection de fautes), l'intégrité du *bitstream* chargé sur le FPGA doit être continuellement vérifiée depuis la mise en marche du système jusqu'à son extinction. Habituellement, les constructeurs de FPGA qui implémentent un module matériel de reconfiguration partielle dans leurs produits permettent aussi la relecture (*Readback*) des *bitstreams* déjà chargés. Cette fonctionnalité peut parfaitement être utilisée pour vérifier en permanence l'intégrité de la configuration. C'est ce que propose par exemple Heiner et al dans [HeCW08] où la configuration du FPGA est régulièrement parcourue grâce à l'ICAP afin de vérifier son intégrité.

Quoiqu'il en soit, les composants FPGA et plus particulièrement la *reconfiguration partielle* rendent possible la mise au point d'architectures *performantes*, *interopérables* et *évolutives* répondant ainsi aux besoins de la radio logicielle sécurisée. Pour cela, un module de reconfiguration sécurisée dédié à la radio logicielle doit proposer les fonctionnalités suivantes : compression, mise en cache, chiffrement et authentification des bitstreams chargés ainsi que la vérification de l'intégrité des bitstreams *déjà* chargés. La Figure 25 illustre de façon simplifiée (le module de contrôle n'est pas représenté) la structure d'un tel

composant. A l'exception de module de reconfiguration qui est nécessairement une IP matérielle, les autres modules peuvent être implantés sous forme d'IP logicielle. Le module cryptographique sert au déchiffrement et à l'authentification des bitstreams envoyés à l'ICAP. La mémoire cache embarquée dans le FPGA sert au stockage des bitstreams déchiffrés et de leur condensé, elle permet un accès rapide à ces données. Le module de reconfiguration est quant à lui un module présent en dur dans le FPGA, il permet un accès en lecture/écriture à la matrice de configuration du FPGA. Pour finir, le module de contrôle d'intégrité vérifie régulièrement l'intégrité du bitstream chargé dans la matrice de configuration du FPGA en utilisant pour cela une fonction de hachage cryptographique et les condensés des bitstreams stockés dans la mémoire cache.

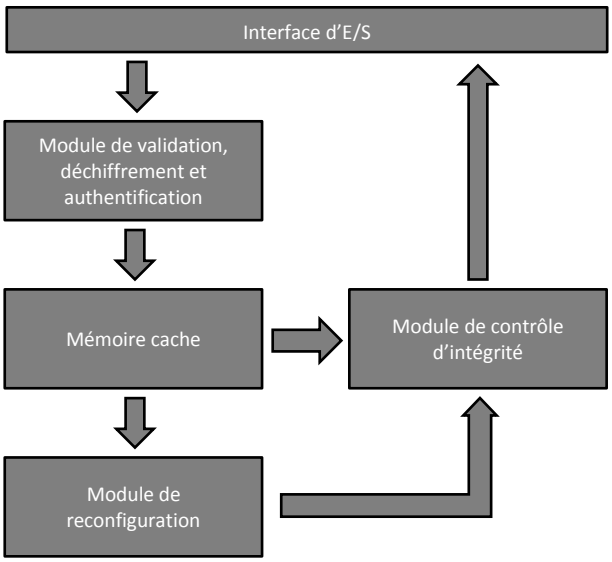


Figure 25 : Schéma synthétique d'un ICAP sécurisé

Plusieurs solutions publiées dans la littérature, ou disponibles dans les circuits FPGA modernes, permettent déjà de répondre à ces besoins. Afin de réduire les temps de développement, nous avons fait l'hypothèse, dans le cadre de cette thèse, que nous disposions déjà d'une interface d'auto-reconfiguration sécurisée et fiabilisée.

3.2.2 **Isolation physique des parties matérielles**

En ce qui concerne les applications militaires, le niveau de sécurité d'une architecture est un facteur déterminant. Le chapitre précédent a montré que l'aspect sécurité n'était pas forcément pris en compte lors de la conception d'architectures dédiées à la cryptographie. En général, ces architectures stockent les clés cryptographiques dans les registres généraux des processeurs ou dans la mémoire vive non sécurisée du système, ce qui les rend vulnérables aux attaques logicielles. En fait, il est possible d'atteindre un meilleur niveau de sécurité en concevant des architectures intrinsèquement sûres en utilisant, par exemple, une méthodologie de conception consistant à séparer un système en deux zones (zone rouge et

zone noire) isolées l'une de l'autre. Dans ce cas, le partitionnement en deux zones (cf. Figure 6) est mise en place aussi bien au niveau fonctionnel que matériel afin de réduire la probabilité d'une fuite d'information par des canaux cachés. De plus, le chemin de données permettant le transfert des clés cryptographiques doit être conçu de manière à rendre impossible la fuite de clés cryptographiques par les interfaces d'E/S du MCCP.

Malheureusement, la structure à base réseaux d'interconnexions des FPGA rend difficile la mise en place d'une isolation électrique au sein de celui-ci bien qu'elle soit nécessaire à l'obtention d'un haut niveau de sécurité [McMM07]. Jusqu'à récemment, les FPGA n'étaient donc pas adaptés à l'implantation d'interfaces cryptographiques entre les zones rouge et noire des systèmes de communication militaire. En 2007, Huffmire et al. ont présenté les premiers résultats concernant la possibilité de réaliser une isolation électrique de différentes parties d'un circuit implanté dans les FPGA [HBWS07]. L'année suivante, la société Xilinx en collaboration avec la NSA présentait une solution logicielle propriétaire baptisée Xilinx *Single Chip Crypto* destinée à répondre à ce problème. Cette solution se compose d'une version modifiée de l'outil Xilinx ISE avec des algorithmes de routage spécifiques et d'un outil de vérification nommé *IVT* pour *Isolation Verification Tool* chargé de vérifier la bonne isolation des différentes parties formant le circuit. Il est à noter qu'un flot de conception équivalent existe aussi depuis 2009 chez Altera sous l'appellation *Quartus II Design Separation Flow* [Alte09].

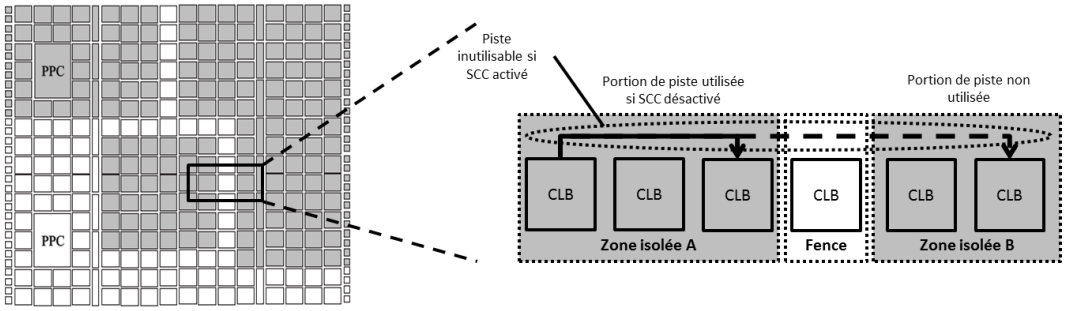
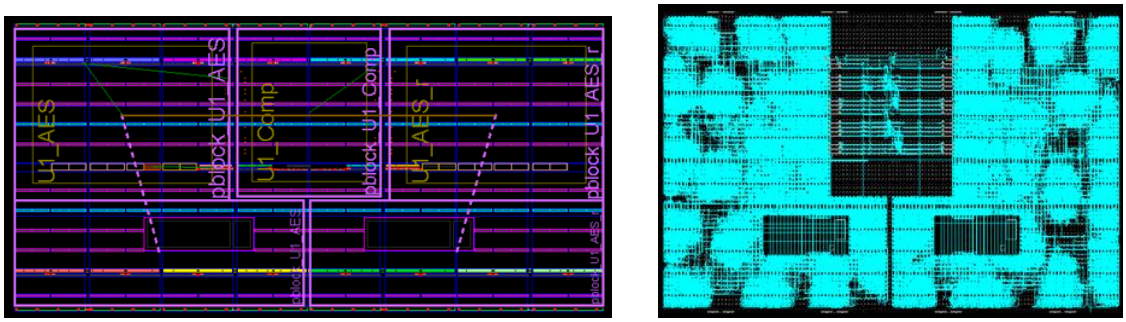


Figure 26 : Isolation avec Xilinx *Single Chip Crypto*



a) Définition des zones isolées sous Xilinx Plan Ahead

b) Floorplan après placement et routage sur Xilinx FPGA Editor

Figure 27: Vues représentant l'isolation de différentes parties d'un circuit électronique

La Figure 26 illustre la proposition d'isolation physique faite par Xilinx pour la famille de circuit Virtex. Les parties isolées représentées en gris sur cette figure sont séparées par des blocs d'éléments logiques vierge (appelés CLB pour *Configurable Logic Block*) dont l'utilisation est interdite. Le « coût » de l'isolation d'un circuit mesuré en nombre de CLB est donc fonction de la taille et du nombre de zones isolées dans un circuit. Les CLB qui font partie de la zone d'isolation sont nommées CLB *fence*. A l'exception des signaux d'horloge utilisés pour synchroniser le circuit, aucune piste ne doit être routée si elle traverse une *fence*. On peut voir sur la partie droite de la Figure 26 qu'une piste qui traverse plusieurs parties isolées, c'est-à-dire une piste qui traverse un CLB *fence*, ne peut *pas* être utilisée. Lors du placement et du routage, l'outil de routage *n'utilise pas* les pistes traversant deux zones isolées. Pour finir, chaque zone possède son propre banc d'entrées-sorties avec sa propre ligne d'alimentation. La solution SCC est compatible avec le niveau de certification « Type 1 » définit par la NSA pour les circuits classifiés (niveau de sécurité le plus bas).

Nous avons mis en œuvre le flot d'isolation proposé par Xilinx sur un composant Virtex-II Pro. Le circuit ciblé a trois parties, deux cœurs d'AES et un comparateur qui doivent être isolés physiquement. La Figure 27 présente l'utilisation du flot de conception Xilinx Single Chip Crypto. La Figure 27.a présente le floorplan utilisé pour décrire les contraintes de placement et d'isolation des trois zones (U1_AES, U1_Comp et U1_AES_r) avec l'outil Xilinx Plan Ahead. La Figure 27.b présente le placement routage final obtenu après utilisation du flot de conception SCC (floorplan obtenu à l'aide de l'outil Xilinx FPGA editor).

Le fait que les flots d'isolation d'Altera et Xilinx aient été validés par la NSA et qu'ils soient en outre disponibles et utilisables font d'eux de bons candidats en ce qui concerne les mécanismes de sécurisation des radios logicielles implantées sur FPGA. En revanche, ces fabricants étant américain, il n'est pas évident que l'Union Européenne puisse comme la NSA avoir accès aux mécanismes internes de ces outils. Et, s'il semble que les solutions d'Altera et Xilinx soient suffisamment abouties pour être utilisées dans les systèmes militaires nécessitant un faible niveau de sécurité, encore faut-il qu'elles soient formellement validées par nos gouvernements. Quoiqu'il en soit l'existence commerciale de ces flots d'isolation montre que le sujet a déjà fait l'objet d'études intensives et nous ne reviendrons donc pas dessus dans la suite de ce document.

Les problématiques de configuration et de sécurité spécifiques aux circuits FPGA ayant été abordées, la suite de ce chapitre traitera exclusivement des problématiques liées à la conception d'un cryptoprocesseur multicœur reconfigurable.

3 Principes architecturaux et interfaçage d'un MCCP

3.3.1 Description de l'architecture

Dans la partie précédente, nous avons présenté quelques-uns des concepts et techniques mis en œuvre lors de la conception l'architecture MCCP que nous proposons. Dans cette

sous-partie, l'architecture MCCP (cf. Figure 28) est présentée. L'intégration du MCCP dans le *bloc de communication* d'un *Crypto Sub System* est, quant à elle, discutée dans la sous-partie suivante. Sur la Figure 28, les flèches noires verticales représentent le flot de données qui traverse le MCCP de haut en bas. Les flèches noires horizontales partant de la mémoire de clé décrivent les chemins de distribution des clés. Enfin les flèches grises tracées en pointillés représentent les signaux de contrôle internes à l'architecture.

On retrouve sur cette figure l'idée d'une architecture multicœur et multicanal dotée de N entrées et N sorties permettant de relier de façon transparente le MCCP à 2*N processeurs ou interfaces réseaux différents. L'avantage d'une telle architecture réside dans sa capacité à répartir de façon globale sur chacun des cœurs la charge réseau provenant des différents ports de communication. Pour cela, des composants appelés *crossbar* ou encore *commutateur* assurent la jonction entre les différents ports d'E/S et les cœurs cryptographiques. Alors que le *crossbar* de *sortie* est composé exclusivement d'éléments combinatoires, le *crossbar d'entrée* embarque une mémoire tampon permettant le stockage des paquets de données mis en attente par l'ordonnanceur de tâches.

Le *contrôleur principal* du MCCP est chargé de la gestion et de la configuration des différents éléments composant le cryptoprocesseur, mais aussi de l'ordonnancement du traitement des paquets de données entrants. Ce composant est contrôlé depuis l'extérieur par le biais de *l'interface de contrôle* du MCCP et reçoit, par exemple, les ordres d'ouverture et de fermeture des canaux de communication. Le *contrôleur principal* ne gère pas directement le processus de reconfiguration partielle du MCCP. Cette tâche est attribuée au module de reconfiguration dynamique partielle présenté dans la partie 3.2.1 et représenté sur la Figure 28 par le bloc ICAP. Plus précisément, chaque bitstream chargé dans l'ICAP (via un port spécifique de *l'interface de contrôle*) se voit attribuer un identifiant qui lui est propre. Cette identifiant est ensuite utilisé par le *contrôleur principal* pour spécifier quel bitstream doit être chargé lors d'un processus de reconfiguration partielle. Ce type de fonctionnement permet de garantir le cloisonnement du système dans la mesure où aucun composant du MCCP hormis l'ICAP n'a directement accès aux bitstreams partiels.

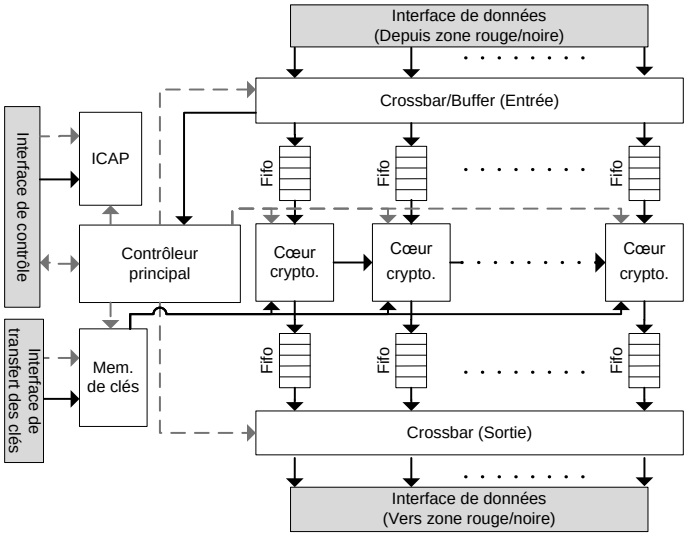


Figure 28: Schéma simplifié de l'architecture d'un MCCP

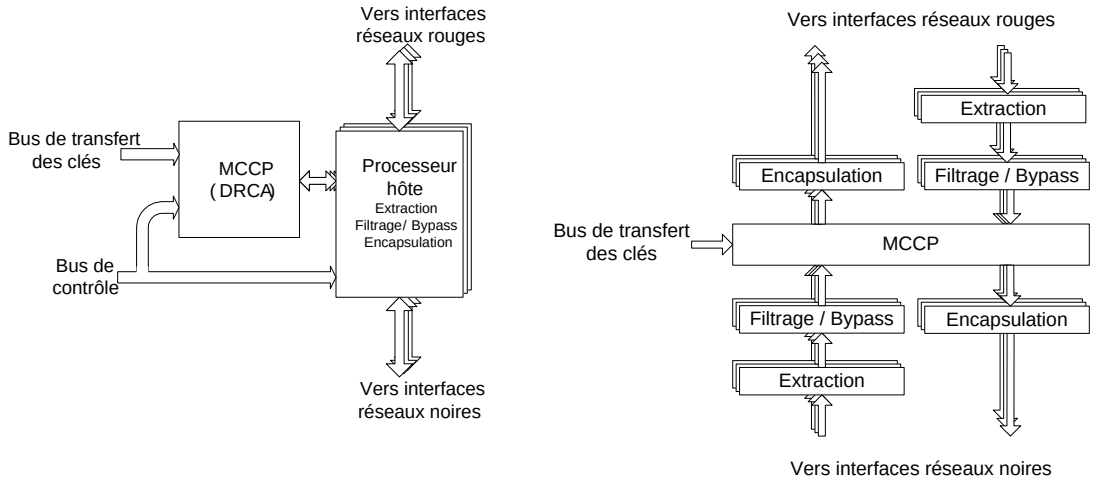
Les cœurs cryptographiques sont les composants centraux du MCCP, car ce sont eux qui assurent la fourniture des services cryptographiques. L'architecture du MCCP est conçue de manière à pouvoir théoriquement embarquer un nombre quelconque de ces cœurs. Afin qu'il soit aisé d'ajouter ou de retirer des cœurs cryptographiques au MCCP, son architecture est basée sur un parallélisme à couplage faible. Autrement dit, chaque cœur fonctionne indépendamment des autres. L'avantage d'une telle structure réside dans sa capacité à s'adapter aux besoins des applications. Les processus de traitement de paquets sont parfaitement adaptés aux architectures parallèles à couplage faible dans la mesure où les paquets sont indépendants les uns des autres.

Outre le fait qu'une telle architecture puisse facilement être étendue, les architectures parallèles à couplage faible ont bien d'autres avantages. D'abord, le fait que les cœurs soient isolés permet d'améliorer le niveau de fiabilité et de sécurité de l'ensemble du système. Il s'ensuit que la défaillance d'un cœur n'est en rien une panne critique du système qui pourra continuer à fonctionner en mode dégradé. Ensuite, ce couplage faible permet d'assurer matériellement l'isolation des processus de traitement des paquets et par la même de réduire le risque de fuite d'informations. De plus, l'isolation physique des cœurs grâce aux outils Xilinx ou Altera présentés précédemment permet d'augmenter encore le niveau de sécurité. Pour finir, l'isolation des cœurs fait que la reconfiguration partielle de l'un d'eux n'a pas d'impact sur le fonctionnement des autres.

Chaque cœur cryptographique est doté de ses propres FIFO d'entrée et de sortie qui servent à stocker le contenu d'un paquet en cours de traitement. Cet aspect de l'architecture est traité plus en profondeur dans la suite de ce chapitre (voir partie 3.4.2).

3.3.2 **Environnement de fonctionnement d'un MCCP**

Un cryptoprocesseur de type MCCP peut être intégré de deux manières différentes au sein d'une radio logicielle. La Figure 29 illustre les deux cas de figure possibles. Sur la Figure 29.a, on retrouve l'architecture présentée en page 41 à la Figure 9. Le MCCP joue ici le rôle du DRCA intégré au *bloc de communication* et son ou ses processeurs hôtes jouent le rôle du processeur généraliste du *bloc de communication*. Chaque processeur hôte se comporte comme une passerelle entre les parties rouge et noire du sous-système cryptographique en faisant transiter les paquets de données à chiffrer/déchiffrer par le MCCP. Ce type de système basé sur une architecture processeur/coprocesseur a l'avantage d'être matériellement simple à mettre en œuvre. La contrepartie étant que la complexité du système est reportée sur le logiciel exécuté par le processeur hôte. En effet, si l'on combine la gestion des canaux de communication à la prise en charge des étapes du traitement d'un paquet de données (extraction, filtrage, encapsulation) alors la charge maximum du système (en nombre de canaux ou en débit) peut être rapidement atteinte. Une solution à ce problème consiste à utiliser un ou des processeurs hôtes plus puissants. Mais, cette solution n'est pas tenable sur le long terme. Une alternative consiste à prendre exemple sur les architectures de *processeurs réseaux*.



a) Bloc de communication basé sur une architecture « processeur/coprocesseur »

b) Bloc de communication basée sur une architecture « processeur réseau »

Figure 29: Différents modes d'implantation du MCCP dans un CSS

La Figure 29.b schématise l'architecture d'un *bloc de communication* de CSS basé sur le principe des *processeurs réseaux,* principe qui est décrit dans le livre de Ran Giladi [Gila08] qui traite ce sujet de façon exhaustive. Chaque bloc de cette figure représente une couche de processeurs réseaux qui exécutent une unique étape du processus de traitement d'un paquet. Chaque bloc est connecté à un bus de contrôle **non indiqué sur la figure** qui est lui-même connecté au *bloc de contrôle* du CSS. Les *processeurs réseaux* sont conçus de manière à

fournir un support matériel à la plupart des traitements appliqués aux paquets transitant sur un réseau. Ils embarquent notamment des modules permettant de trier, d'extraire et de comparer différents types de données ou encore de rechercher au travers d'un arbre un jeu de données (ex : adresse IP).

L'indépendance de chacun des cœurs cryptographiques embarqués dans le MCCP fait qu'il peut aisément être inséré au sein d'une telle architecture. Avec ce genre de configuration, le flot de données est habituellement unidirectionnel. Toutefois, un même MCCP peut traiter aussi bien des paquets entrants que des paquets sortants et il est tout à fait possible d'envisager la gestion d'un flux bidirectionnel par le MCCP. Néanmoins, l'impact d'une telle décision doit être soigneusement mesuré. Car, dans ce cas, aucun mécanisme matériel n'empêche qu'un paquet destiné à la zone rouge se trouve réorienté vers la zone noire. La solution la plus sécurisée est donc de prévoir des chemins de données distincts pour chacun des modes de fonctionnement (i.e. rouge/rouge, noir/noir, rouge/noir, noir/rouge) en attribuant de façon statique un certain nombre de cœurs cryptographiques à chacun d'eux.

Contrairement au cas représenté sur la Figure 29.a, les données qui ne nécessitent pas l'application d'opérations cryptographiques transitent quand même par le MCCP ce qui rend nécessaire l'ouverture d'un canal neutre sur celui-ci. Le concepteur a alors deux options : soit il choisit d'ajouter au MCCP un chemin de données permettant de contourner les cœurs cryptographiques, soit il décide de configurer les cœurs de manière à ce qu'ils n'appliquent aucune transformation aux paquets de données transitant par le canal neutre.

3.3.3 **Interfaçage d'un cryptoprocesseur basé sur l'architecture MCCP**

L'architecture du MCCP dispose de quatre interfaces de communication que l'on retrouve sur la Figure 28 : une interface de contrôle, deux interfaces d'E/S destinées au transfert des paquets de données et une interface de transfert des clés. Chacune de ces interfaces à un rôle et un fonctionnement bien distinct. On remarquera que l'interface de transfert des clés, qui est isolé du reste du MCCP, est directement connectée à la mémoire de stockage des clés.

L'interface de contrôle :

L'interface de contrôle permet la gestion des cryptoprocesseurs basés sur l'architecture MCCP, elle permet notamment d'ouvrir ou fermer un canal et de contrôler le statut du cryptoprocesseur (ex. tentative d'intrusion, échec d'authentification, …). Elle est connectée au processeur du *Bloc de Contrôle* du CSS par le biais d'une liaison asynchrone bufferisée permettant l'échange de messages entre le *Bloc de Contrôle* et le cryptoprocesseur MCCP. A cet effet, l'interface de contrôle est équipée en entrée et en sortie de FIFO destinées à stocker les messages de contrôle entrant et sortant. L'avantage principal d'un tel type de liaison réside dans sa capacité à limiter les échanges entre le *Bloc de Contrôle* et le cryptoprocesseur au strict minimum. Un second avantage réside dans la possibilité d'envoyer au *Bloc de Contrôle* des *messages de statut* l'informant de l'état du cryptoprocesseur

sans qu'il n'y ait eu, au préalable, de sollicitation de sa part. Une telle fonctionnalité est particulièrement utile lorsqu'il s'agit de notifier l'occurrence d'une intrusion ou d'un échec d'authentification. Par contre, ce type de liaison nécessite que le *Bloc de Contrôle* fasse correspondre entre eux les messages entrant et sortant. Pour cela, le *contrôleur principal* du MCCP répond aux messages du *Bloc de Contrôle* dans leur ordre de réception. Quant aux *messages de statut* ceux-ci sont identifiés de manière à ce qu'ils ne soient pas confondus avec les *messages de réponse*. Les messages transitant aux travers de l'interface de contrôle peuvent être classifiés selon le Tableau 6.

Tableau 6: Classification des messages transitant par l'interface de contrôle

Appellation	Signification	Destination du message
OPEN	Messages demandant l'ouverture d'un canal	Cryptoprocesseur
CLOSE	Messages demandant la fermeture d'un canal	
STATUS	Messages de demande de statut	
ROPEN	Réponse à une demande d'ouverture d'un canal	Bloc de contrôle
RCLOSE	Réponse à une demande de fermeture d'un canal	
RSTATUS	Réponse à une demande de statut	
EVENT	Message informant de l'occurrence d'un évènement particulier	

Tableau 7: Format des messages de l'interface de contrôle

Type de message	Contenu
OPEN	• Identifiant d'algorithme • Taille du tag d'authentification • Identifiant de clé • Taille du vecteur d'initialisation (32, 64, 96 ou 128bits) • Vecteur d'initialisation • Niveau de priorité (optionnel)
CLOSE	• Identifiant de canal
STATUS	• Identifiant de registre de statut
ROPEN	• OK ou code d'erreur • Identifiant de session (si OK)
RCLOSE	• OK ou code d'erreur
RSTATUS	• OK ou code d'erreur • Valeur demandée (si OK)
EVENT	• Code d'évènement • Canal concerné (optionnel)

Chaque message est constitué d'un champ identifiant le type du message puis d'un contenu qui varie en fonction de ce type. Le

Tableau 7 spécifie le contenu de chacun des différents types de message. L'encodage des messages n'est pas présenté ici. En effet, celui-ci dépend des spécifications de l'architecture du MCCP à implanter. En outre, il est tout à fait possible d'ajouter de nouveaux types de message à ceux préexistants.

Les codes d'erreur des messages de type *ROPEN*, *RCLOSE* ou *RSTATUS* signifient au *Bloc de Contrôle* l'apparition d'une erreur lors du traitement d'un message entrant. On peut citer comme erreur : un identifiant de clé incorrect ou le fait que la limite du cryptoprocesseur MCCP en nombre de ressources utilisées est atteinte. Dans le cas où l'identifiant de message serait mal formaté, un message de type *EVENT* est envoyé au *Bloc de Contrôle*. De façon plus générale, les messages de type *EVENT* permettent d'informer le *Bloc de Contrôle* de l'occurrence d'un évènement quelconque. Les messages de type *EVENT* peuvent être utilisés pour les raisons, non exhaustives, suivantes :

- Une instruction ou un paquet mal formaté a été reçu.
- L'authentification d'un paquet de données a échoué.
- Une tentative d'intrusion a été détectée.
- La validité dans le temps d'un vecteur d'initialisation ou d'une clé est arrivée à son terme.

En parallèle de l'envoi d'un message de type EVENT, le MCCP peut aussi lancer l'exécution d'une routine quelconque. Par exemple, lorsque l'authentification d'un paquet échoue, le *supplément sécurité à la SCA* spécifie que le paquet doit être rejeté et que l'utilisateur doit être informé [Jtrs04]. De plus, si un certain nombre d'erreurs est atteint, le canal doit être automatiquement fermé.

ID_C	ID_M	MODE	S_AAD	S_PT	Données AAD	Données PT	TAG

Figure 30: Format d'un paquet de données à chiffrer/déchiffrer

Les interfaces de données :

Les interfaces de données permettant le transit des paquets ont, quant à elles, un fonctionnement différent. Comme expliqué précédemment, chaque paquet (cf. Figure 30) est marqué avec l'identifiant du canal (ID_C) auquel il appartient. Il peut aussi être optionnellement marqué avec un identifiant de message (ID_M) afin de permettre son suivi tout au long des différents modules de traitement qui composent la forme d'onde. En outre, l'en-tête du paquet contient aussi un champ indiquant si le paquet doit être chiffré ou déchiffré (MODE) ainsi que deux autres spécifiant la taille des données qui doivent être seulement authentifiées (S_AAD) et celle des données qui doivent être à la fois authentifiées et chiffrées (S_PT). Pour finir, lorsque le paquet est destiné à être déchiffré ou qu'il vient d'être chiffré, un champ TAG est présent à la fin du paquet. Ce dernier contient le condensé nécessaire à l'authentification du paquet. La Figure 30 permet de visualiser le format d'un paquet de donné entrant. Sur cette figure la taille en bits de chacun des champs n'est pas spécifiée, car cette caractéristique peut varier en fonction des besoins.

Comme pour l'interface de contrôle, le processus de communication au niveau des interfaces de données du MCCP repose sur le modèle de la FIFO. Au niveau du *crossbar d'entrée*, chaque port dispose d'une interface constituée d'un bus de données, d'un signal d'écriture et d'un signal indiquant si le port est saturé ou non. Le *crossbar de sortie* dispose de l'interface de sortie habituelle d'une FIFO. C'est-à-dire d'un bus de données, d'un signal de lecture et d'un signal spécifiant si la queue est vide. La Figure 31 illustre cette construction.

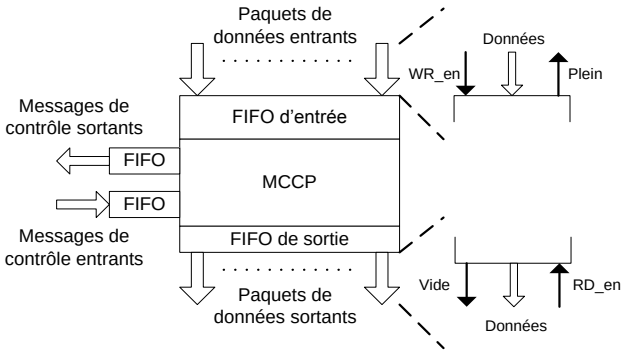


Figure 31: Modélisation des interfaces de communication d'un MCCP

3.4 Fonctionnement d'un MCCP

3.4.1 Le contrôleur principal

Alors que la section précédente présente de façon générale l'architecture du MCCP, cette partie s'attache à détailler le fonctionnement des divers éléments de contrôle et de glue logique nécessaires à son fonctionnement. Si l'on met de côté les cœurs cryptographiques dont l'architecture est détaillée dans la section suivante, l'architecture du MCCP est principalement composée de quatre éléments, qui sont : le *contrôleur principal*, les *crossbars,* la *mémoire de stockage des clés* et le module de reconfiguration partielle ICAP (*Internal Configuration Access Port).*

Le *contrôleur principal* est un élément central de l'architecture du MCCP. Les tâches suivantes lui sont attribuées :

- Gestion de l'interface de contrôle et des canaux.
- Gestion et configuration des cœurs cryptographiques.
- Ordonnancement des paquets entrants.
- Gestion de la distribution des paquets aux cœurs cryptographiques.

L'attribution de l'ensemble de ces tâches au *contrôleur principal* permet d'une part de décharger significativement le reste du système de tâches qui peuvent être consommatrices en temps machine. D'autre part, cela permet d'améliorer l'isolation fonctionnelle entre les différents éléments constituants le système. Le nombre des tâches attribuées au *contrôleur principal* et leur caractère asynchrone font qu'il n'est pas aisé de développer un logiciel

adapté. Un système d'exploitation simple, mais multitâche pouvant alors être nécessaire. Afin de clarifier le rôle du *contrôleur principal*, la suite de cette section détaille les étapes nécessaires au traitement d'un paquet de données. En faisant l'hypothèse qu'un ou plusieurs canaux de communication ont déjà été ouverts sur le MCCP, le traitement par le *contrôleur principal* des paquets entrants se fait de la manière suivante (voir la Figure 32 pour une représentation graphique des différents processus, les étapes correspondantes sont notées entre guillemets) :

1. Les paquets entrants sont stockés dans le *tampon* par le *crossbar d'entrée*. Comme précisé précédemment, chaque paquet est marqué par l'identifiant du canal auquel il appartient. Le contrôleur principal est notifié de l'arrivée d'un nouveau paquet par l'intermédiaire d'une file de messages.
2. Le contrôleur vide la file de messages et considère que les paquets qui sont liés à ces messages sont arrivés au cours de la même *fenêtre temporelle*. La liste des paquets en attente est alors mise à jour, chaque item de la liste est horodaté de façon à préciser sa *fenêtre temporelle d'arrivée*. Cet horodatage peut, par la suite, être utilisé par des algorithmes d'ordonnancement tel qu'EDF (*Early Deadline First*).
3. Tant que la liste des paquets entrants n'est pas vide, le *contrôleur principal* effectue les actions suivantes :
    a. À partir de la liste des paquets en attente et de la configuration courante des cœurs cryptographiques, il détermine le prochain paquet à traiter et le cœur cryptographique qui effectuera le traitement. La configuration courante des cœurs peut avoir un impact significatif sur le choix du prochain paquet à traiter et du cœur qui le traitera. En effet, les performances d'un cœur peuvent varier en fonction de sa configuration et du type d'algorithme cryptographique à exécuter.
    b. Le *contrôleur principal* autorise le *crossbar* à transférer le paquet de données sélectionné vers le cœur sélectionné.
    c. Le *contrôleur* configure le cœur cryptographique chargé du traitement du paquet. Cette configuration peut nécessiter l'envoi d'une clé de chiffrement ou d'un vecteur d'initialisation au cœur cryptographique si celui-ci ne les possède pas déjà en cache. Il peut aussi être nécessaire de reconfigurer partiellement le cœur sélectionné par l'intermédiaire de l'ICAP. Le contrôleur principal n'a alors qu'à spécifier le cœur qui doit être reconfiguré et l'algorithme qui doit y être chargé, l'ICAP se charge du reste. Pour finir, le contrôleur spécifie le traitement (c.-à-d. l'algorithme cryptographique) qui doit être appliqué au paquet de données avant d'autoriser le cœur à commencer le traitement du paquet.
4. Lorsque l'un des paquets a été complètement traité, le *contrôleur principal* se le voit notifié. Il autorise alors le transfert du paquet traité vers un port de sortie donné et considère que le cœur cryptographique est à nouveau libre. Si l'authentification du paquet a échoué (ce qui provoque la destruction du paquet), le *contrôleur principal*

incrémente un compteur et lorsqu'une valeur seuil est atteinte un message *EVENT* est envoyé au *Bloc de Contrôle*.

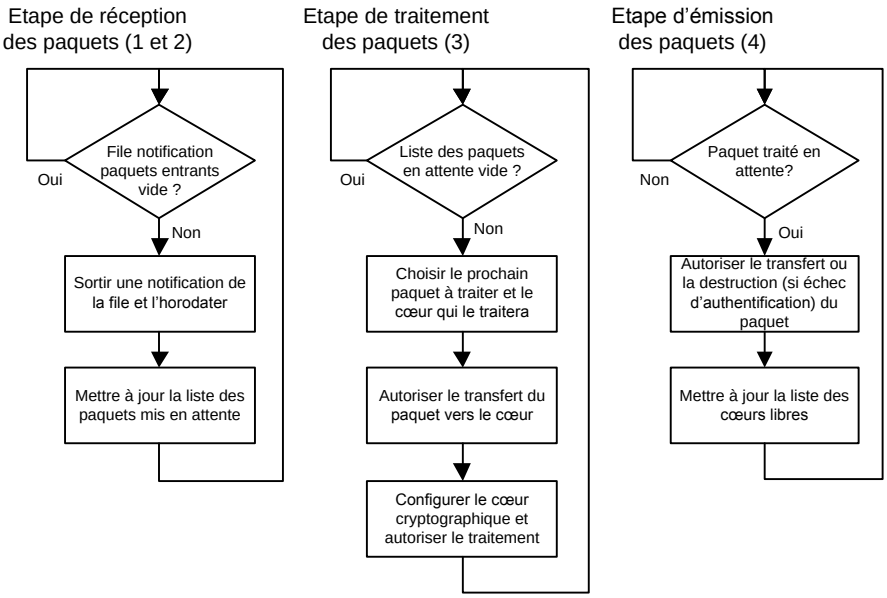


Figure 32 : Processus de contrôle mis en œuvre par le contrôleur principal

Le détail du processus de traitement montre clairement que le *contrôleur principal* n'intervient que très peu dans le traitement des paquets. Son rôle se borne à la gestion des tâches de plus haut niveau. Bien que cela puisse sembler peu, il n'en est rien. En effet, les processus d'ordonnancement de tâches lorsqu'ils ne sont pas triviaux peuvent être consommateurs en temps machine. Il s'agit donc de limiter autant que possible les charges incombant au *contrôleur principal* lorsque la puissance de celui-ci est limitée. Le *contrôleur principal* peut être implanté de plusieurs façons selon les besoins du système. Dans le chapitre suivant, un simple microcontrôleur 8 bits est utilisé dans le cadre d'un ordonnancement de type FIFO[14]. Pour l'implantation d'un algorithme d'ordonnancement plus complexe, l'utilisation d'un processeur 16 voire 32 bits peut être indispensable.

3.4.2 **Interfaces de communication du MCCP**

Comme nous l'avons dit, la gestion du transfert des paquets au niveau des différentes interfaces est assurée conjointement par les cœurs cryptographiques et les *crossbars* d'entrée et de sortie. Alors que le *crossbar* de sortie permet uniquement le transfert d'un paquet depuis n'importe quel cœur cryptographique vers n'importe quel port de sortie, le *crossbar*

[14] Les paquets sont traités dans leur ordre d'arrivée.

d'entrée permet en outre de stocker les paquets en attente de traitement. L'implantation d'un *crossbar* n'est pas forcément triviale dans la mesure où celui-ci est constitué de N entrées et M sorties ce qui, implanté sous forme de multiplexeur, peut nécessiter un grand nombre de portes logiques. Une solution alternative permettant de diminuer le coût en surface silicium des crossbars réside dans l'utilisation de réseaux d'interconnexions.

Il existe plusieurs topologies de réseaux d'interconnexions qui sont couramment utilisés pour l'implantation de commutateurs réseaux, on pourra citer par exemple, les réseaux de Clos [Clos53] ou encore ceux de Benes [RhMi90]. Ces derniers, illustré sur la Figure 33 ont plusieurs avantages. D'une part ce type de réseau peut facilement être étendu pour un coût supplémentaire limité[15] et d'autre part, il est résistant aux fautes. En effet, il existe dans un réseau de Benes à N entrées, N/2 chemins de données différents pour chaque couple de ports entrée/sortie. Toutefois, le fait que ce type de crossbar soit symétrique (c.-à-d. qu'il possède N entrées et N sorties) est problématique dans la mesure où il est souhaitable de pouvoir faire varier le nombre de cœurs indépendamment du nombre d'entrées. Par ailleurs, la configuration d'un réseau de Benes peut poser problème lorsque celui-ci est de grande taille. Ce problème peut néanmoins être résolu en calculant à l'avance les tables de routages ou en intégrant des mécanismes d'auto-routage au réseau [NaSa81].

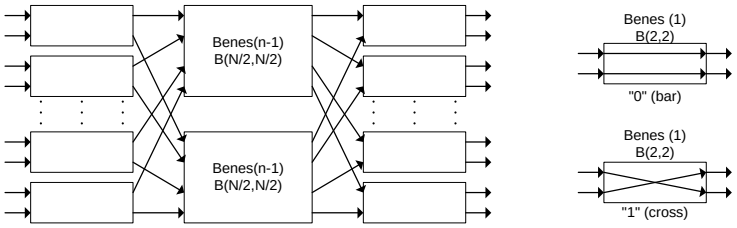


Figure 33: Réseau d'interconnexion de Benes de taille NxN avec $N=2^n$

En ce qui concerne le *crossbar d'entrée*, le problème est différent, car celui-ci embarque une mémoire tampon à laquelle tous les ports d'entrée et de sortie du *crossbar* doivent être reliés. Il existe plusieurs classes d'architectures de queue qui permettent de répondre au besoin du *crossbar d'entrée*. La plupart d'entre elles sont présentées dans [KaVE95] et illustrées en Figure 34. Le Tableau 8 récapitule leurs avantages et inconvénients. L'implantation sur FPGA des architectures basées sur les structures suivantes : *Double Internal Switch*, *Output Queueing*, *Shared Buffer* et *Block-Crosspoint* pose de sérieux problèmes. En effet, de par la structure interne des FPGA (constitué de blocs RAM ayant au plus deux ports d'entrée), les mémoires partagées et les FIFO à N ports d'entrée ne sont pas des ressources dont l'implantation est envisageable voir même possible.

[15] Complexité en nombre d'élément B(2,2) en O(n*ln(n))

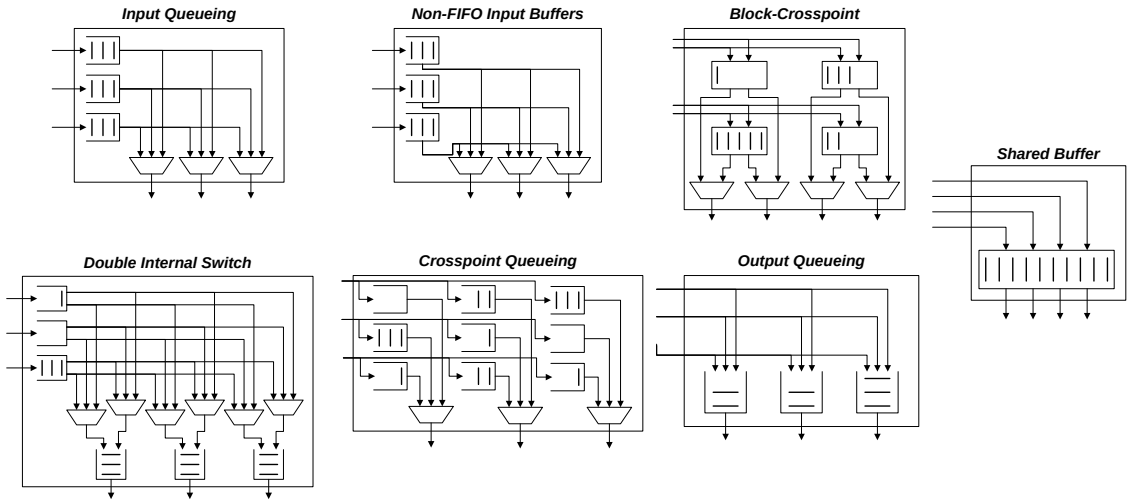


Figure 34 : Architectures de queue d'entrée [KaVE95]

L'autre inconvénient de la plupart des architectures présentées réside dans l'impossibilité de réordonnancer les paquets en provenance d'un même port d'entrée. Or, il se peut qu'il faille réordonnancer les paquets en provenance d'un même port d'entrée pour des raisons de QdS. D'ailleurs, le NIST explique clairement dans [KuWF05] que le traitement des paquets dans leur ordre d'arrivée pose de sérieux problèmes de QdS dans les applications de VoIP sécurisées. Les seules architectures présentées en Figure 34 qui permettent de traiter les paquets en attente dans un ordre quelconques sont les architectures *Shared Buffer, Block-Crosspoint* implantées à l'aide de RAM et l'architecture *Non-FIFO Input Buffer* qui permet un accès aléatoire aux mots stockés dans les buffers d'entrée. Or, nous avons déjà fait remarquer qu'il est difficile et coûteux d'implanter sur FPGA des mémoires partagées à N ports. L'architecture *Non-FIFO Input Buffers* est donc la seule architecture qui permette l'ordonnancement des paquets tout en conservant un coût d'implantation raisonnable dans la mesure où de simples blocs RAM peuvent être utilisés comme buffers d'entrée. Une autre approche envisageable pourrait consister à implanter une architecture *Crosspoint Queuing* basée sur des buffers identiques à ceux de l'architecture *Non-FIFO Input Buffers*.

Tableau 8 : Comparatif de différentes architectures de crossbar bufferisé

Architecture	Débit	Avantages	Inconvénient
Input Queuing	Bas	• Implantation simple.	• Lorsqu'un paquet est mis en attente tous les autres paquets de la file sont aussi mis en attente (*head of line blocking*).
Non-FIFO Input Buffer	Bas	• Règle le problème de *head of line blocking*.	• Nécessite l'utilisation d'un algorithme d'ordonnancement complexe. • Impossible de rediriger en même temps des paquets issus d'une même file.
Double Internal Switch	Bas	• Améliore le débit en sortie en fonctionnant à une fréquence supérieur à celle des liens d'entrée et de sortie.	• Les files doivent être équipées de deux ports d'entrée. • Le réseau d'interconnexions est complexe.
Crosspoint Queuing	Elevé	• Permet une utilisation optimale du débit disponible au niveau des ports de sortie.	• Nécessite une quantité importante de mémoire.
Output Queuing	Elevé	• Améliore l'utilisation mémoire par rapport au *Crosspoint Queuing*.	• Nécessite l'implantation de FIFO à N entrées.
Shared Buffer	Elevé	• Utilisation plus efficace de la mémoire pour un débit comparable aux architectures *Output Queuing et Crosspoint Queuing*.	• Nécessite de gérer la fragmentation de la mémoire. • Mémoire complexe à implanter.
Block-Crosspoint	Elevé	• Réduit la contrainte de débit sur les mémoires partagées par rapport au *Shared Buffer*. • Meilleure utilisation de la mémoire que le *Crosspoint Queuing*.	• Nécessite de gérer la fragmentation de la mémoire. • Mémoire complexe à implanter. • Moins performant que le *Shared Buffer*.

Dans un souci de simplicité et d'économie des ressources mémoires, nous considérerons que les crossbars du MCCP sont basées sur une architecture *Non-FIFO Input Buffers*. L'architecture du *crossbar d'entrée* prendrait alors la forme illustrée en Figure 35. Sur cette figure, on retrouve en haut à droite, les buffers d'entrée qui permettent de lire les paquets dans un ordre quelconque. En bas à droite, on retrouve le réseau d'interconnexions qui peut être construit à partir de simples multiplexeurs ou de structures plus complexes (ex. réseaux d'interconnections de Clos ou de Benes). A gauche de la figure, on trouve le contrôleur du *crossbar*. Celui-ci a deux rôles : D'une part, il doit notifier au contrôleur principal l'arrivée de nouveaux paquets. D'autre part, il doit contrôler les buffers et le réseau d'interconnexions en fonction des notifications entrantes qui concernent la distribution des paquets aux différents cœurs. Les mécanismes de communication entre le *contrôleur principal* et le *crossbar d'entrée* sont implantés sous forme de FIFO afin d'en simplifier le fonctionnement.

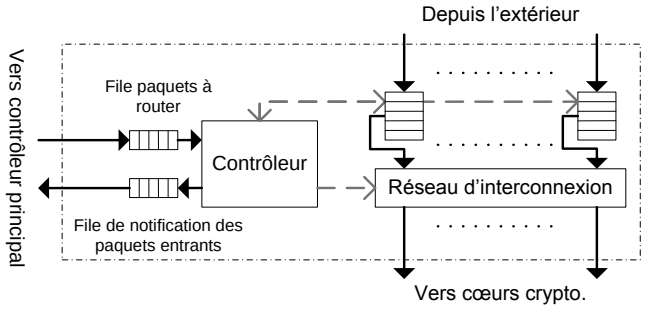


Figure 35 : Architecture du crossbar d'entrée

Pour finir, notons que la structure des FPGA n'est pas adaptée à l'implantation d'un crossbar bufferisé. D'abord, la mise au point de mémoires à N ports d'entrées indépendants sur FPGA n'est pas une chose aisée, car ceux-ci embarquent des mémoires RAM équipées de seulement deux ports indépendants. Ensuite, les réseaux de Benes largement utilisés pour la fabrication de commutateurs et crossbar sur ASIC ne sont pas adaptés à une implantation sur FPGA. En effet, un nœud élémentaire d'un réseau de Benes possède trois entrées (deux pour les données et une pour l'adressage) et deux sorties or les FPGA modernes (Virtex 5 et 6) embarquent des LUT à cinq entrées et deux sorties. L'implantation de crossbars sous la forme de réseaux de Benes conduit donc à une utilisation inefficace de la surface silicium disponibles. En fait, la structure des FPGA actuels ne permet pas une implantation *efficace* des crossbars qu'ils soient bufferisés ou non.

Plus généralement, on peut se demander si la structure actuelle des FPGA est réellement adaptée à l'implantation de systèmes multicœur. Dans une première approche, on pourra remarquer que la surface actuelle des FPGA permet l'implantation d'un MCCP composé d'une trentaine de cœurs cryptographiques (si l'on se réfère aux résultats d'implantation du Chapitre 4). Mais que penser d'un système embarquant un plus grand nombre de cœurs de calcul ? Deux solutions s'offrent à nous : soit la structure des FPGA est modifiée de façon à embarquer un nombre toujours plus grand de composants spécifiques (ex. mémoires à N ports, crossbars, etc.), soit on suit l'approche inverse et on décide d'intégrer dans un ASIC un ou plusieurs composants FPGA. On parle alors d'eFPGA (*embedded FPGA*). L'étude de cette approche pourrait servir de base à de futurs travaux de recherche.

3.5 Les cœurs cryptographiques

3.5.1 Evaluation des besoins : Etudes des algorithmes et modes de chiffrement

Les cœurs cryptographiques sont les unités de traitement d'un MCCP. Ils ont pour rôle le chiffrement et le déchiffrement des paquets de données ainsi que leur authentification. L'architecture MCCP est conçue de manière à ce que les cœurs cryptographiques soient aussi autonomes que possible. À cet effet, chaque cœur doit embarquer un certain nombre de mécanismes qui doivent permettre : le chiffrement/déchiffrement des paquets, leur authentification, leur mise en forme et la communication avec le reste du système. La

conception de l'architecture des cœurs cryptographiques passe donc par la définition des besoins en termes d'opérateurs cryptographiques et de mécanismes de communication.

Dans le domaine militaire, la sécurité par le secret est un concept largement répandu et bien que l'architecture présentée dans ce document soit en partie destinée aux applications militaires, nous nous limiterons ici aux algorithmes cryptographiques civils pour des raisons de confidentialité. Par ailleurs, afin de limiter l'étendue de ces travaux, nous nous restreignons au cas des algorithmes de chiffrement symétriques. À l'heure actuelle, l'algorithme AES [Nist01a] fait office de standard de chiffrement dans le domaine civil. De plus, le Tableau 9 nous montre que l'algorithme AES est quasi systématiquement utilisé par les protocoles de communication courants. C'est donc lui qui a été choisi pour illustrer le fonctionnement de notre architecture. Il existe toutefois, un grand nombre d'algorithmes reconnus sûrs (au moins pour les applications civiles) par les cryptologues. Parmi ceux-ci, on pourra citer Serpent ou encore Twofish. A contrario, il existe d'autres algorithmes considérés comme peu sûrs mais qui sont encore largement utilisés, notamment DES, IDEA ou encore RC4. Dans ce document, nous nous limiterons aux algorithmes de chiffrement par bloc dont les caractéristiques correspondent aux conditions définies par le NIST lors de la compétition qui mena à l'adoption du nouveau standard AES. C'est-à-dire, une taille de bloc de 128 bits et des tailles de clé de 128, 192 et 256 bits. Plus d'informations sur la compétition du NIST et l'implantation des finalistes AES sont disponibles dans [EYCP01].

Tableau 9 : Algorithmes cryptographiques utilisés par différents protocoles de communication

Protocole de communication	Algorithmes cryptographiques mis en œuvre
WIFI (WPA2)	AES
MACsec	AES
IPsec	3DES, AES, SHA
TLS	AES, 3DES, SHA, RSA
Wimax	AES
Zigbee	AES

En ce qui concerne l'implantation des algorithmes de chiffrement par bloc, Elbirt et al. ont montré qu'un petit nombre d'opérateurs mathématiques simples permet de répondre aux besoins de la grande majorité d'entre eux [ElPa03]. En effet, seul huit opérateurs différents (cf. Tableau 10) sont nécessaires à l'implantation d'une quarantaine d'algorithmes cryptographiques à clé privée[16]. Par ailleurs, on peut remarquer que certain de ces opérateurs peuvent encore être décomposés en opérateurs de plus bas niveau (ex. : multiplication dans les corps de Galois $GF(2^n)$ qui peut être décomposé en plusieurs XOR et opérations de décalage). *L'interopérabilité* d'une architecture *multimode* (c.-à-d. qui supporte

[16] Blowfish, CAST, CAST-128,CAST-256, CRYPTON, CS-Cipher, DEAL, DES, DFC, E2, FEAL, FROG, GOST, Hasty Pudding, ICE, IDEA,Khafre, Khufu, LOKI91, LOKI97, Lucifer, MacGuffin, MAGENTA, MARS, MISTY1, MISTY2, MMB, RC2,RC5, RC6, Rijndael, SAFER K, SAFER+, Serpent, SQUARE, SHARK, SKIPJACK, TEA, Twofish, WAKE, et WiderWake

plusieurs algorithmes) dépend donc directement du nombre et du degré d'atomicité des opérateurs mathématiques qu'elle embarque. Malheureusement, l'atomicité de ces opérateurs est à l'origine des faibles performances en termes de débit et/ou de latence de ces architectures : un grand nombre d'opérations atomiques sont nécessaires au chiffrement d'un bloc de données. On comprend donc que plus la granularité d'une architecture *multimode* est faible plus le nombre de cycles d'horloge nécessaires au chiffrement d'un bloc de données est élevé. D'un autre côté, il n'est pas techniquement viable d'implanter dans un composant cryptographique quarante et un blocs de chiffrement dédiés dans le but de le rendre aussi interopérable et performant (en terme de débit) que possible. C'est donc au concepteur de trouver un compromis adapté à ses besoins. Les deux solutions extrêmes à ce problème étant une architecture *multimode* ayant **une faible granularité** ou une **juxtaposition** de *composants dédiés*.

Tableau 10 : Occurrence des opérateurs parmi les algorithmes cryptographiques à clé secrète

Opérateur	Occurrences (sur 41)
Booléen	40
Addition et soustraction modulaire	20
Décalage fixe	25
Rotation variable	10
Multiplication modulaire	7
Multiplication dans les corps de Galois	7
Inversion modulaire	1
Table de substitution	30

Pour des raisons de sécurité, les algorithmes de chiffrement par bloc ne sont presque jamais utilisés seuls, car, pour une clé donnée, un bloc de données sera toujours chiffré de la même façon. Ce problème peut être résolu en utilisant les modes de chiffrement validés et publiés par le NIST (cf. Annexe C). Ces modes[17] de chiffrement sont pour la plupart basés sur l'utilisation conjointe d'une primitive cryptographique, de XOR, de compteurs et d'autres opérateurs mathématiques. Ceux-ci sont communs à tous les algorithmes de chiffrement symétrique par bloc. Les modes de chiffrement des algorithmes de chiffrement par blocs se divisent en trois classes englobant des modes de *chiffrement*, *d'authentification* et de *chiffrement authentifié*. Le Tableau 11 illustre cette classification.

Tableau 11: Classification des modes de chiffrement

Classes de mode de chiffrement	Modes de chiffrement du NIST	Spécifications correspondantes
Chiffrement	ECB, CBC, CFB, OFB et CTR	SP 800-38A
Authentification	CBC-MAC, OMAC et XCBC	SP 800-38B
Chiffrement authentifié	CCM et GCM	SP 800-38C et D

[17] CBC, CTR, OFB, CCM, GCM, etc.

Le Tableau 12 présente les modes de chiffrement mis en œuvre par différents protocoles de communication. Celui-ci montre que seuls quelques modes de chiffrement sont communément utilisés dans le domaine des communications. Dans la suite de ce document, nous considérerons donc exclusivement les modes ECB, CTR, CBC, CBC-MAC, CCM et GCM[18].

Tableau 12: Modes de chiffrement utilisés par différents protocoles

Protocole de communication	Mode de chiffrement
WIFI (WPA2)	CCM
MACsec	GCM
IPsec	CBC et HMAC[19]
TLS	CBC et Signature SHA/RSA
Wimax	CCM
Zigbee	CCM

Le chemin de données correspondant à l'implantation d'un mode de chiffrement peut être divisé en trois parties. La première correspond à la primitive cryptographique de chiffrement (ex. AES), la seconde correspond à la primitive cryptographique d'authentification (ex. multiplieur de Galois ou AES) et la dernière fait office de glue logique entre les deux premières. Le tout formant le mode de chiffrement. Le Tableau 13 détaille les opérateurs nécessaires au fonctionnement de chacun des modes de chiffrement précédemment sélectionnés. Celui-ci montre que les modes de chiffrement sont basés sur des glues logiques comparables.

Tableau 13: Opérateur mis en œuvre par les modes de chiffrement sélectionnés

Modes	Opérateurs mis en œuvre		
	Primitive de chiffrement	Primitive d'authentification	Glue logique
CTR	AES	Aucune	XOR, incrémentation
CBC	AES	Aucune	XOR
CBC-MAC	Aucun	AES	XOR, comparateur
CCM	AES	AES	XOR, incrémentation, comparateur
GCM	AES	Multiplication dans un corps de Galois	XOR, incrémentation, comparateur

Que ce soit pour l'implantation des algorithmes de chiffrement ou celle des modes de chiffrement, le concepteur d'un crypto-système interopérable doit faire un compromis entre une architecture multimodes et une architecture à base de composants dédiés. En réalité, il est possible de dépasser ce compromis, notamment en utilisant la *reconfiguration dynamique partielle* des FPGA SRAM présentée précédemment.

[18] cf. Annexe C pour une description des modes de chiffrement.
[19] Hash-based Message Authentication Code

3.5.2 Application de la reconfiguration dynamique partielle aux cœurs d'un MCCP

Grâce à la *reconfiguration dynamique partielle*, il est maintenant possible de reconfigurer une partie d'un circuit en fonction des besoins. Un crypto-système n'a donc plus besoin d'être compatible dès sa conception avec l'ensemble des algorithmes cryptographiques, que ce soit en implantant une architecture multimode ou en juxtaposant un grand nombre de composants dédiés. Toutefois, le délai nécessaire à la reconfiguration partielle d'un FPGA fait que le changement de contexte provoqué par le passage d'un algorithme à un autre pénalise fortement les performances générales du système. Il s'ensuit que d'une part, la zone reconfigurée doit être aussi petite que possible et que d'autre part, le circuit doit être reconfiguré le moins souvent possible. On en vient donc à un problème de conception où il s'agit de déterminer la granularité idéale de la zone reconfigurable.

La partie précédente a montré que l'algorithme de chiffrement symétrique utilisé varie peu suivant le protocole de communication utilisé (voir Tableau 9). Au contraire des modes de chiffrement qui eux varient d'avantage en fonction des protocoles de communications (voir Tableau 12). Par ailleurs, les modes de chiffrement sont basés sur des structures simples (cf. Annexe C). Il semble donc pertinent de réduire le périmètre des zones reconfigurables *partiellement* à celui des primitives cryptographiques de chiffrement et d'authentification. Tandis que la glue logique des modes de chiffrement peut être avantageusement implantée dans une zone reconfigurable *statique* sous la forme d'une architecture multimode. De cette façon, le passage d'un protocole de communication à un autre peut se faire très rapidement lorsqu'ils sont basés sur le même algorithme de chiffrement. Par contre, les changements de protocole de communication qui supposent un changement de l'algorithme de chiffrement (cas plus rares si l'on suit les informations des Tableau 9 et Tableau 12) sont plus longs dans la mesure où une reconfiguration partielle du circuit est nécessaire. Nous voyons bien ici l'intérêt de profiter à la fois des avantages d'une structure programmable (passage rapide d'un mode à un mode) et de ceux d'une structure reconfigurable partiellement (optimisation de la surface nécessaire).

En outre, l'utilisation d'une architecture multicœur permet de réduire fortement le nombre de reconfigurations partielles nécessaires au fonctionnement du système. En effet, chaque cœur peut être configuré avec des primitives cryptographiques différentes de manière à privilégier *l'interopérabilité* en supprimant le délai nécessaire au passage d'une primitive à une autre. A contrario, les cœurs peuvent être tous configurés avec la même primitive afin d'augmenter le débit global du système lorsqu'un algorithme particulier est utilisé sur tous les canaux. Evidemment, une configuration intermédiaire consistant à attribuer un certain nombre de cœurs à l'exécution d'un algorithme et le reste à l'exécution d'un autre est tout à fait envisageable. De manière générale, les architectures multicœur tirent profit de la reconfiguration partielle dans le sens où celle-ci permet le passage d'une architecture multicœur homogène à une architecture multicœur hétérogène selon les besoins. De plus, pour une charge réseau donnée, plus le nombre de cœurs est important moins les délais de

reconfiguration seront pénalisants. En effet, il devient alors plus facile de transférer la charge de travail d'un cœur vers d'autres tout en limitant l'impact sur les performances.

Dans la partie suivante, nous proposons une architecture de cœur cryptographique programmable et dynamiquement reconfigurable basée sur l'ensemble des conclusions tirées précédemment.

3.5.3 **Architecture et fonctionnement des cœurs cryptographiques**

L'étude précédente conduit à la réalisation de l'architecture de cœur cryptographique illustrée sur la Figure 36. Sur celle-ci, la structure globale de l'architecture est de type Harvard possédant des mémoires de données et d'instructions différentes. Nous ne traiterons pas ici de la largeur des bus des chemins de données des cœurs cryptographiques : le choix de celle-ci est laissé au concepteur. Toutefois, la solution la plus adaptée semble d'adopter une largeur de bus de 32 ou 64 bits pour l'ensemble du cœur à l'exception des primitives cryptographiques dont la largeur de bus devrait dépendre de l'algorithme (soit 32, 64 ou 128 bits).

Concernant le fonctionnement des interfaces d'E/S, selon le mode de fonctionnement choisi, le cœur peut soit lire les données d'entrées depuis la FIFO d'entrée, soit les lire depuis le cœur de gauche (cf. Figure 36). De la même façon, l'écriture des données de sortie peut se faire à destination de la FIFO de sortie ou à destination du cœur de droite. Les ports de communications directes inter-cœur permettent de répartir la charge de traitement d'un même paquet sur plusieurs cœurs si cela s'avère nécessaire. Lorsque cette fonctionnalité est utilisée, le flot de données traverse l'ensemble des cœurs concernés en partant du cœur d'entrée (à gauche) et en allant jusqu'au cœur de sortie (à droite). Toutefois, l'approche consistant à distribuer le traitement de différents paquets d'un même canal sur différents cœurs est à privilégier dans la mesure où cette dernière facilite la gestion des ressources par le *contrôleur principal*.

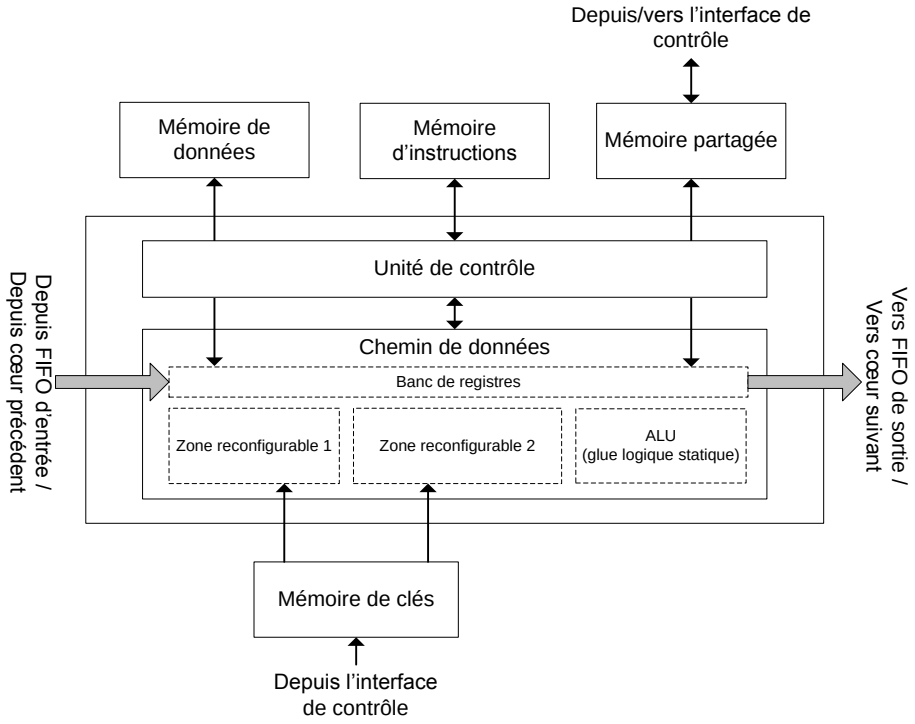


Figure 36 : Architecture d'un cœur cryptographique reconfigurable d'un MCCP

Chaque cœur est divisé en deux unités fonctionnelles : une *unité de contrôle* et un *chemin de donnéess*. L'*unité de contrôle* permet le décodage des instructions, la gestion du pointeur d'instruction et la configuration du *chemin de donnéess*. Les cœurs cryptographiques communiquent avec le *contrôleur principal* par l'intermédiaire d'une mémoire partagée. L'implantation d'un mécanisme de gestion des interruptions dans les cœurs cryptographiques est facultative dans la mesure où il est possible de faire du *polling* sur les interfaces d'E/S sachant que chaque cœur n'a jamais plus d'une seule tâche à effectuer à la fois. En ce qui concerne le *chemin de donnéess* des cœurs, il est composé : d'un banc de registres, d'une ALU embarquant les opérateurs logiques et arithmétiques courants ainsi que de deux zones reconfigurables. L'ALU fournit les opérateurs nécessaires à la glue logique des modes de chiffrement et au fonctionnement du cœur cryptographique.

Les zones reconfigurables, toutes deux identiques, sont destinées à l'implantation des primitives cryptographiques (ex. AES, Twofish, multiplieur de Galois, SHA, Whirlpool, etc.). Les primitives de chiffrement et d'authentification peuvent être indifféremment configurées dans chacune des deux zones reconfigurables de manière à optimiser l'utilisation des ressources matérielles disponibles. Afin de permettre l'utilisation concurrente des primitives configurées dans les zones reconfigurables, ces dernières sont équipées d'accès indépendants et strictement isolées à la mémoire de clés. Dans le cas général, chaque primitive cryptographique de chiffrement (c.-à-d. AES, Twofish, etc.) doit embarquer son propre module de génération des sous-clés cryptographiques. Néanmoins,

lorsque l'architecture ne supporte qu'un seul type de primitive cryptographique de chiffrement (ex. AES), ce module peut être partagé par l'ensemble des cœurs cryptographiques et être intégré au système en amont des mémoires de clés de chaque cœur. Le processus de traitement des paquets varie selon le mode de chiffrement sélectionné et la configuration des cœurs. Toutefois, celui-ci suit toujours le schéma général illustré par la Figure 37.

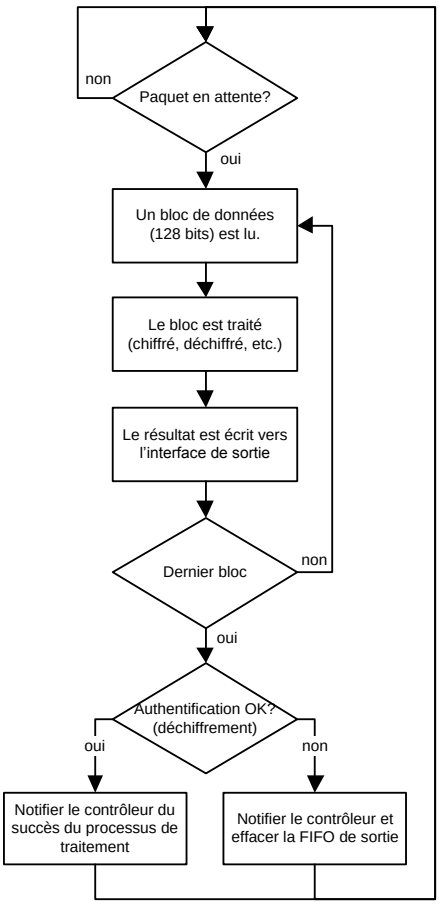


Figure 37 : Processus de traitement d'un paquet par un cœur cryptographique

Les performances des cœurs cryptographiques dépendent principalement du type des primitives configurées dans les zones reconfigurables. Certaines configurations peuvent être plus efficaces que d'autres lorsqu'il s'agit d'exécuter un mode de chiffrement donné. D'un autre côté, le passage d'une configuration à une autre peut introduire des délais gênants. Le Tableau 14 illustre les performances relatives de différentes configurations. Le

débit de référence est noté « 1x », un débit deux fois plus grand est noté « 2x » et un débit deux fois moins grand « 0,5x ». On considère ici que les deux primitives cryptographiques disponibles sont un bloc de chiffrement AES et un multiplieur de Galois et que la primitive la plus lente est le bloc de chiffrement AES. Bien sûr, ce tableau peut évoluer en fonction du type des primitives cryptographiques disponibles. Celui-ci montre que la configuration AES/AES fournit de bonne performance en général mais n'est pas compatible avec GCM alors que la configuration AES/Mul. de Galois est compatible avec l'ensemble des modes au détriment des performances.

Tableau 14: Débit relatif en fonction de la configuration des zones reconfigurables

Mode de chiffrement	Configurations (zone 1 / zone 2)	
	AES / AES	AES / Multiplieur de Galois
ECB	2x	1x
CTR	2x	1x
CBC-MAC	1x	1x
CCM	1x	0,5x
GCM	Incompatible	1x

3.6 Ordonnancement des paquets

3.6.1 Description de la problématique

Si l'on considère que les performances d'un réseau sont limitées, il s'ensuit que celui-ci doit opérer un arbitrage afin d'utiliser de façon optimale les ressources disponibles. C'est à cette problématique d'arbitrage que répondent les algorithmes d'ordonnancement. Concrètement, les algorithmes d'ordonnancement proposent différentes manières de régler les conflits d'accès aux ressources matérielles quand un réseau est congestionné [CuRe95]. C'est-à-dire, quel paquet doit être traité en premier et quel autre en second lorsque l'on souhaite maintenir une *qualité de service* globale satisfaisante.

Il existe de nombreuses méthodes d'ordonnancement dont l'intérêt dépend de la structure du réseau, de la *qualité de service* attendue, de la complexité de leur implantation et de leurs performances. D'après [Gila08], les méthodes d'ordonnancement peuvent être classifiées de différentes façons. Une façon de les départager [RaPa03] est de considérer d'un côté les algorithmes basés sur la notion d'échéance et les autres basés sur le tour par tour. Les algorithmes basés sur le temps et la notion d'échéance sont particulièrement bien adaptés aux applications temps réel dans la mesure où ils permettent un contrôle précis des temps de latence. Mais, ils peuvent souffrir d'une complexité calculatoire importante. Au contraire, les algorithmes basés sur le tour par tour (*Round Robin*) sont simples à implanter mais ils ne permettent pas un contrôle précis des temps de latence. Une autre façon de classifier les méthodes d'ordonnancement [Hui95] est de considérer celles qui sont à *travail conservatif* et celles qui ne les sont pas. Les méthodes d'ordonnancement à *travail conservatif* effectuent leurs traitements tant qu'il y a des paquets en attente. A l'inverse les méthodes d'ordonnancement *non conservatives* peuvent temporairement interrompre le processus d'ordonnancement. Ces dernières sont utiles lorsqu'il est nécessaire de limiter la gigue des

paquets en sortie de l'ordonnanceur. Toutes ces classes de méthodes d'ordonnancement ont fait l'objet d'importants travaux, nous invitons le lecteur à se reporter à [Gila08] pour d'avantage d'informations.

Les « performances » d'une méthode d'ordonnancement peuvent être évaluées selon plusieurs critères parmi lesquels on retrouve, les délais, le taux d'utilisation des ressources, la complexité, la robustesse et l'équité. Il se trouve que la plupart de ces critères sont le résultat de compromis où l'amélioration de l'un se fait au détriment d'un autre. En général, le contrôle des délais prime sur le reste des critères d'évaluation, vient ensuite la complexité puis l'équité de traitement entre les canaux. Concernant l'architecture MCCP, le fait qu'elle soit destinée entre autres au chiffrement de flux audio ou vidéo (dans le cadre de l'audio ou de la vidéo conférence) amène à privilégier le contrôle des délais par rapport aux autres critères de performance.

Figure 38 : Ordonnancement mono-tâche et multitâche

A l'origine, de nombreuses méthodes d'ordonnancements étaient destinées aux systèmes de traitement mono-cœur. Avec cette approche, l'ordonnancement de paquets consiste à lire des paquets depuis N files entrantes et à les copier sur la sortie de l'ordonnanceur. L'ordre dans lequel sont copiés les paquets dépend de la méthode d'ordonnancement utilisée. Par la suite, les systèmes multicœur sont apparus. Cela a conduit à généraliser la notion d'algorithme d'ordonnancement en considérant N sorties. La Figure 38 schématise ces deux approches.

L'architecture MCCP étant multicœur, elle est basée sur la seconde approche. A chaque sortie de l'ordonnanceur correspond un cœur cryptographique. Quant aux files d'entrée, elles ne correspondent pas forcément à un port d'entrée. Nous avons vu au cours de la section 3.4.1 que le *crossbar d'entrée* notifie au *contrôleur principal* l'arrivée et la nature des paquets entrants. Le *contrôleur principal* a donc la possibilité de répartir les paquets entrants en différentes files virtuelles qui sont par exemple fonction de la nature des paquets ou de leur canal d'appartenance. Ainsi, si l'on considère qu'à chaque file d'entrée correspond un canal de communication, il est possible de rendre certains canaux prioritaires par rapport à d'autres.

L'étude des techniques d'ordonnancement sur l'architecture MCCP est une problématique complexe (problème d'optimisation NP-difficile) qui pourrait faire l'objet d'un chapitre complet de thèse. Les travaux présentés dans cette section ne cherchent donc pas à donner une réponse exhaustive à cette problématique. Par contre, il nous a semblé indispensable de

définir précisément les problèmes d'ordonnancement relatifs au MCCP puis de proposer quelques pistes de réflexions quant à la manière de les résoudre.

De façon générale, il est possible de définir la problématique d'ordonnancement des paquets par le MCCP de la manière suivante : Il s'agit d'exécuter un ensemble de *tâches* au moyen d'un ensemble de *ressources* tout en remplissant un certain nombre d'*objectifs*. Les *tâches, ressources* et *objectifs* sont définis de la façon suivante :

- **Les *tâches* :** Les tâches à exécuter prennent la forme de paquets à traiter. Chaque paquet appartient à un canal et un canal est caractérisé par :
  - Un débit qui doit être garanti.
  - Une latence maximale.
  - Le type de traitement à appliquer aux paquets du canal.
  - Une taille de paquet éventuellement variable.
- **Les *ressources* :** Les *ressources* correspondent aux *cœurs cryptographiques*, elles ont les propriétés suivantes :
  - Le traitement d'un paquet n'est pas interruptible.
  - Le traitement d'un paquet ne peut pas être transféré d'une unité à une autre.
  - Les unités de calcul ont des performances différentes selon le type de paquet et certaines sont incapables de traiter certains types de paquets.
- **Les *objectifs* :**
  - Chaque canal doit recevoir la bande passante qui lui est requise.
  - La latence maximum des canaux doit être respectée.
  - La charge doit être équitablement répartie sur les différentes unités de calcul.
  - Les paquets d'un même canal doivent être émis dans l'ordre où ils sont reçus (contrainte souple, d'autres composants du réseau peuvent réordonnancer les paquets au prix d'une latence et d'une gigue accrues).

Le problème d'ordonnancement étant maintenant posé, la partie suivante peut introduire quelques pistes de réflexion sur la manière de le résoudre.

3.6.2 Quelques pistes de réflexion

Tout d'abord, l'ordonnancement de paquets étant un problème NP-difficile, il doit être résolu grâce à une approche heuristique. Cette partie présente donc tour à tour deux heuristiques d'ordonnancement ayant un intérêt certain dans le cadre de nos travaux.

Le MCCP peut être amené à traiter des types de paquets divers et variés. Chaque type de paquet possède des contraintes de qualité de service différentes. Par exemple, la qualité de transmission des paquets audio et vidéo dépend fortement de la latence des communications et quelques pertes de données sont acceptables. Il n'en est pas de même des transmissions de données textuelles dont l'intégrité doit être préservée. Il s'ensuit qu'appliquer un même algorithme d'ordonnancement à tous les types de paquets ne semble pas être une stratégie efficace tant au niveau de l'efficacité de l'ordonnancement en lui-

même qu'au niveau de la gestion des ressources de calcul nécessaires à l'exécution du processus d'ordonnancement. Dans [WoGa03], Ganz et al. proposent d'utiliser des algorithmes d'ordonnancement différents selon le type des paquets traités en suivant une approche d'ordonnancement hiérarchique. Ils proposent dans un premier temps d'ordonnancer les paquets par ordre de priorité, chaque type de paquet (fichiers de données, flux temps réel, etc.) possédant son propre niveau de priorité. Puis, dans un second temps, d'utiliser des algorithmes adaptés aux types des paquets à ordonnancer (ex. EDF (*Earliest Deadline First*) pour les paquets contenant des données issues de flux temps réel). Cette approche a le mérite de tenir compte de la diversité des contenus transitant sur un réseau. En outre, elle permet de simplifier le processus d'ordonnancement dans la mesure où elle vise à diviser la tâche d'ordonnancement en plusieurs problèmes de moindre complexité. Ce type d'approche semble tout à fait pertinent dans le domaine de la radio logicielle étant donné la diversité des types de flux de données à traiter.

Plus proche du contexte de cette thèse, les travaux de Ferrante et al. [FePC05] traitent de l'ordonnancement du traitement des paquets sur une architecture de crypto-système multicœur dédié au chiffrement de flux IPSec. Leur proposition de mécanismes d'ordonnancement repose sur le principe suivant :

- Dans un premier temps, les paquets sont redirigés en direction de l'un des N cœurs de chiffrement. Pour chaque paquet entrant, le module de redirection calcule le délai nécessaire au traitement du paquet par chacun des cœurs (c.-à-d. temps d'attente dans la file du cœur plus le temps de chiffrement/déchiffrement). Le paquet est redirigé vers le cœur ayant le délai de traitement le plus court.
- Dans un second temps, les paquets redirigés sont insérés dans l'une des six files d'attente placées à l'entrée de chaque cœur. Chaque file d'attente correspond à un niveau de priorité IPSec. Chaque cœur accède ensuite à ces files en suivant un algorithme WFQ (*Waited Faire Queuing*).

Cette méthode d'ordonnancement semble tout à fait naturelle. Mais comme toutes les méthodes d'ordonnancement *en-ligne*, elle possède l'inconvénient suivant : le délai de traitement d'un paquet assigné à un cœur est modifié lorsqu'un nouveau paquet de priorité supérieure est assigné au même cœur. Il s'ensuit que le processus d'allocation des tâches procède à des choix qui peuvent s'avérer non optimaux. Comme solution à ce problème, Ferrante et al. proposent d'utiliser une représentation statistique des flux d'informations traversant le crypto-système. Ils proposent d'évaluer pour chaque cœur le temps de calcul nécessaire au traitement des paquets en attente en prenant en compte celui nécessaire au traitement d'éventuels futurs paquets de plus haute priorité. Cette technique d'ordonnancement est problématique dans la mesure où elle nécessite de nombreux calculs. Une autre approche serait d'inverser les étages d'ordonnancement et d'attribution des paquets aux cœurs. Dans ces conditions, le temps de traitement d'un paquet par un cœur pourrait être connu de manière précise. Mais cette approche a l'inconvénient d'éloigner le

processus d'ordonnancement des cœurs (une FIFO étant inséré entre les deux) ce qui aurait pour conséquence de conduire là aussi à des choix d'ordonnancement sous optimaux.

L'aperçu de la problématique donné par ces travaux montre que la mise au point d'une méthodologie d'ordonnancement du traitement des paquets sur le MCCP n'est pas une chose aisée. En outre, la plupart des travaux sur l'ordonnancement des paquets ne sont pas basés sur la même architecture ni le même contexte que les notre. Il s'ensuit que l'ordonnancement du traitement des paquets par le MCCP pourrait, à l'avenir, faire l'objet d'études innovantes et originales en rupture avec l'état de l'art passé.

3.7 **Conclusion**

Le chapitre d'état de l'art ayant mis à jour certaines lacunes des crypto-systèmes existants, le présent chapitre a détaillé plusieurs solutions permettant de les combler. Ces solutions prennent la forme d'un cryptoprocesseur appelé MCCP dédié au chiffrement de flux de données prenant la forme de canaux de communication. Plus précisément, le concept du MCCP repose sur les principes, mécanismes et technologies suivantes :

- *Une architecture multicœur homogène à couplage faible:* Les paquets de données sur un réseau pouvant être généralement traités indépendamment les uns des autres, le MCCP a été conçu sur la base d'une architecture *multicœur homogène à couplage faible*. Ce type d'architecture permet de paralléliser le traitement des paquets tout en garantissant le partitionnement (et donc la sécurité), la flexibilité ou encore l'adaptabilité (en faisant varier le nombre de cœurs) du MCCP. *L'homogénéité* (tous les cœurs sont identiques même s'il est vrai que leur configuration peut changer) de l'architecture du MCCP permet d'en simplifier l'utilisation.
- *Le principe d'isolation fonctionnelle :* Ce principe a influé la conception du MCCP sur différents plans. On pourra notamment citer l'indépendance des cœurs cryptographiques entre eux, la séparation entre les interfaces *de contrôle*, *de transfert des clés* et *de données* ou encore le haut degré *d'autonomie* du MCCP.
- *La technologie FPGA :* La technologie FPGA permet au MCCP de pouvoir évoluer dans le temps. De plus, l'utilisation de la *reconfiguration dynamique partielle* permet de faire évoluer les fonctionnalités du MCCP même lorsqu'il est en cours de fonctionnement. Pour finir, les mécanismes et outils d'isolation des circuits fournis par Xilinx et Altera permet d'assurer l'isolation des différentes composantes du MCCP bien qu'elles soient implantées sur une même puce FPGA.
- *L'ordonnancement du traitement des paquets :* L'architecture du MCCP rend possible l'ordonnancement des paquets. Mais des travaux complémentaires visant à proposer des algorithmes d'ordonnancement adaptés doivent encore être menés.

La mise en œuvre de ces principes mécanismes et technologies repose sur un certain nombre de contributions originales en ce qui concerne la structure du MCCP, celle de ses cœurs cryptographiques ou encore son fonctionnement. Parmi celles-ci, on pourra citer :

- L'architecture multicœur reconfigurable du MCCP qui n'a actuellement pas d'équivalent dans l'état de l'art à l'exception de *Cryptomaniac* qui n'est toutefois pas reconfigurable.
- La structure des cœurs cryptographiques qui leur permet d'être à la fois programmables et reconfigurables dans le but de fournir le meilleur compromis possible entre flexibilité et performances.
- La logique d'interconnexion des cœurs avec l'interface d'entrée qui permet l'ordonnancement des paquets.
- L'isolation au niveau matériel des mécanismes de contrôle (*contrôleur principal*), de traitement de l'information (*cœurs cryptographiques*), de génération, stockage et transfert des clés secrètes.

Etant donné le large éventail de possibilités d'application offert par la radio logicielle, cette présentation du MCCP est restée aussi générale que possible et très peu de détails d'implantation ont donc été fournis. Il est donc difficile d'estimer quantitativement les performances d'un tel processeur théorique. Face à ce constat, le chapitre suivant introduit un exemple concret d'implantation du MCCP. Le temps alloué au développement et au test étant nécessairement limité au cours d'une thèse, seule une version allégée du MCCP a été implantée.

Chapitre 4

Implantation d'un MCCP et résultats

4.1 Introduction

Dans le chapitre précédent, l'architecture théorique du MCCP a été présentée. Cette architecture est théorique dans la mesure où elle est décrite par un ensemble de principes et techniques servant de point de départ pour l'implantation concrète du MCCP détaillé dans ce chapitre. Alors que le chapitre précédent est basé sur une étude qualitative, ce chapitre présente des résultats quantitatifs permettant d'évaluer les performances d'un MCCP. L'implantation du MCCP se veut aussi générique que possible, cependant nous avons dû faire un certain nombre de choix techniques afin de restreindre la complexité du développement. Ce chapitre est donc basé sur un certain nombre d'hypothèses que nous présenterons dans la section 4.2.1.

L'organisation générale de ce chapitre adopte une approche *top down* et suit la structure suivante : la section 4.2 présente les limitations, la structure générale et le fonctionnement du MCCP implanté. La section 4.3 détaille la façon dont les *cœurs cryptographiques* sont implantés dans le MCCP. Cette partie est suivie par la section 4.4 qui présente la structure des *unités cryptographiques* embarquées dans les *cœurs cryptographiques*. Ensuite, la section 4.5 détaille les résultats d'implantation obtenus aussi bien en termes de ressources occupées que de débit/latence. Puis, la section 4.6 introduit un certain nombre de résultats concernant la reconfiguration dynamique partielle et son utilisation pour le MCCP implanté. Enfin, la section 4.7 revient sur l'état de l'art en comparant, dans le cadre de la radio logicielle, les crypto-systèmes existants au MCCP.

4.2 Implantation d'un MCCP

4.2.1 Environnement de test et hypothèses d'implantation d'un MCCP

Nous avons vu dans le chapitre précédent qu'un MCCP peut être interfacé de différentes manières avec le reste du système dans lequel il est intégré. Dans la suite de ce chapitre, nous considérerons le cas, illustré Figure 39, où le MCCP est utilisé en tant que coprocesseur cryptographique rattaché à un unique processeur hôte. Ce processeur joue à la fois le rôle du *bloc de contrôle* et celui du *bloc de communication* (cf. Chapitre 1). Cette approche simplifie grandement la mise en place d'un banc de test dans la mesure où un seul processeur généraliste est utilisé. Bien évidemment, dans un système réel, le *bloc de contrôle* et le *bloc de communication* devraient être implantés séparément. Le processeur hôte est un

processeur MicroBlaze [Xili11b] de Xilinx qui communique par l'intermédiaire d'une connexion série standard avec le PC de test et par l'intermédiaire d'un lien FSL[20] [Xili10b] avec le MCCP. Le processeur MCCP implanté n'étant pas directement compatible avec la technologie de lien FSL, un module permettant l'interfaçage des *bus de contrôle, de données et de clés* du MCCP avec le lien FSL du MicroBlaze a été développé.

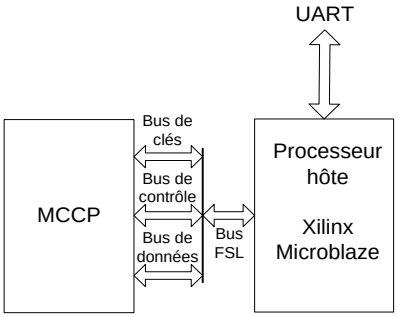


Figure 39 : Plateforme de test du MCCP

Afin de limiter les temps de développement, l'architecture du MCCP implantée et présentée dans ce chapitre est une simplification de l'architecture du MCCP présentée dans la partie précédente. Cette simplification porte principalement sur trois points. D'abord, le *crossbar* d'entrée du MCCP implanté n'inclut pas de buffer d'entrée. C'est donc au processeur hôte de se charger de la mise en tampon et de l'ordonnancement des paquets. Ensuite, le MCCP implanté embarque uniquement un chiffreur déchiffreur de type AES. Pour finir, le mécanisme de reconfiguration partielle est directement contrôlé par le processeur hôte plutôt que par le MCCP. Les hypothèses d'implantation et la plateforme de test ayant été présentées, il est maintenant possible de décrire l'architecture de notre prototype.

4.2.2 **Architecture du MCCP**

Le prototype du MCCP implanté embarque un *contrôleur principal* chargé d'interpréter et d'exécuter les instructions du processeur hôte et de répartir les tâches de traitement des paquets sur les différents cœurs cryptographiques. Le *contrôleur principal* est implanté sous la forme d'un contrôleur 8 bits *Picoblaze* distribué par Xilinx [Ken00]. Différentes versions du Picoblaze ont été utilisées selon le type du FPGA cible (Virtex 4 ou Virtex 6). Comme cela est expliqué dans le chapitre précédent, le *contrôleur principal* est chargé du contrôle des *crossbars* d'entrée et de sortie tous deux non bufferisés, de celui des cœurs cryptographiques et de celui du *générateur de sous clés AES* dont les services sont partagés par l'ensemble des cœurs cryptographiques.

[20] Le lien FSL permet, grâce à des FIFO, d'établir un canal de communication asynchrone entre un processeur MicroBlaze et un coprocesseur.

Le *contrôleur principal* communique avec le processeur hôte par l'intermédiaire du *bus de contrôle* qui permet l'accès à un *registre d'instruction* sur 32 bits et un *registre de retour* sur 8 bits. Les signaux *start, done* et *data_available* sont utilisés pour synchroniser l'exécution des instructions. La Figure 40 illustre la structure de l'architecture du prototype du MCCP présenté dans ce chapitre. Bien que quatre cœurs cryptographiques soient illustrés sur cette figure, le nombre de cœurs cryptographiques n'est pas limité à quatre. L'implantation VHDL actuelle du MCCP permet de faire varier le nombre de cœurs de deux à huit par pas de deux cœurs. Sous réserve de modification du code VHDL, d'avantage de cœurs peuvent être implantés.

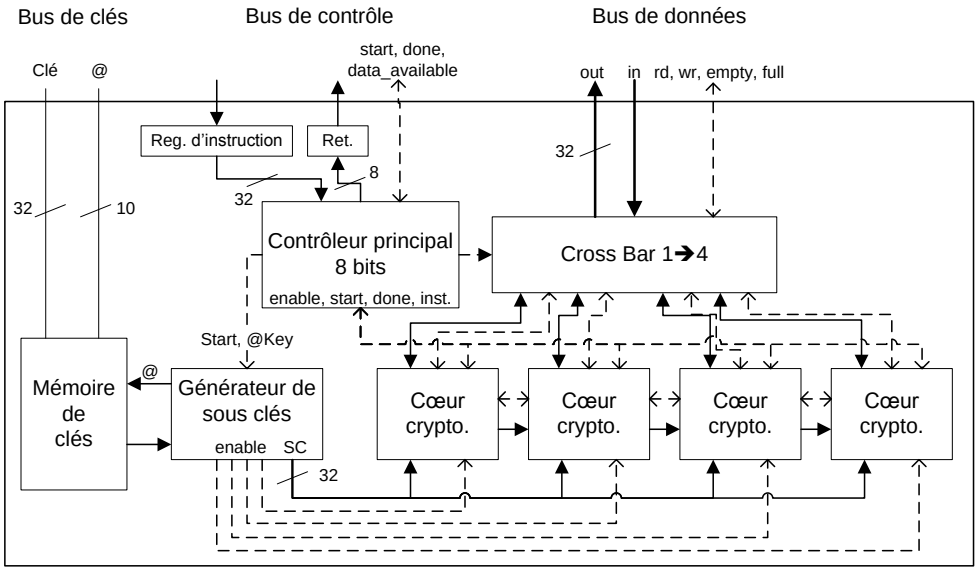


Figure 40 : Architecture du prototype du MCCP

Chaque cœur cryptographique communique avec le processeur hôte par l'intermédiaire du *crossbar*. Celui-ci est implanté sous la forme de multiplexeur (cf. fin de la section 3.4.2). Le *crossbar* a une interface de type FIFO constitué d'un bus de lecture et un autre d'écriture. Le contrôle de l'accès au *crossbar* par le processeur hôte est opéré par le *contrôleur principal*. Le *générateur de sous clés* génère, à partir des clés stockées dans *la mémoire de clés*, les clés de round nécessaires au fonctionnement de l'algorithme AES. Bien que les cœurs cryptographiques se partagent le *générateur de sous clés*, celui-ci est suffisamment performant pour ne pas constituer un goulot d'étranglement pour le MCCP. Il va de soi que si le MCCP doit être compatible avec d'autres algorithmes de chiffrement qu'AES, chaque cœur cryptographique doit être équipé de son propre *générateur de sous clés*. La génération de sous clés cryptographiques suit les étapes suivantes : d'abord le *contrôleur principal* charge l'identifiant de la clé du canal à traiter dans le *générateur de sous clés*, ce dernier se charge alors de récupérer la clé cryptographique dans la mémoire de clé puis de générer et d'écrire les sous-clés dans le cache de clés du cœur de destination. Afin de garantir la protection des

clés, le MCCP n'a pas d'accès en écriture à la *mémoire de clés* et il n'existe aucun mécanisme permettant l'extraction des clés cryptographiques par le port de données du MCCP. L'architecture des cœurs cryptographiques est étudiée plus en détail dans la suite de ce chapitre (cf. partie 3.5).

4.2.3 **Protocole de contrôle du MCCP**

Dans la mesure où le prototype du MCCP implanté n'embarque pas de mémoire tampon dans son *crossbar d'entrée*, le protocole de communication présenté dans le chapitre précédent nécessite une légère modification. Comme cela est expliqué précédemment, le MCCP exécute des instructions 32 bits émises par le processeur hôte et retourne en réponse une valeur codée sur 8 bits. Le jeu d'instructions disponible permet à l'utilisateur d'ouvrir, fermer, chiffrer ou déchiffrer des paquets. En outre, grâce au signal *data_available, le contrôleur principal* peut notifier au processeur hôte la présence de données dans les FIFO de sortie des cœurs cryptographiques. L'exécution d'une instruction par le MCCP suit les quatre étapes ininterruptibles suivantes :

1. Le processeur hôte écrit l'instruction à exécuter dans le *registre d'instruction* du *MCCP*.
2. Le processeur hôte envoie le signal *start* au MCCP.
3. Le processeur hôte attend que le MCCP envoie le signal *done*.
4. Le processeur hôte lit la valeur de retour renvoyée par le MCCP.

Le jeu d'instructions du MCCP est composé des instructions suivantes :

- OPEN *ID_Algorithme, ID_clé* : Cette instruction permet d'ouvrir un nouveau canal de communication sur le MCCP. Elle retourne soit un code *OK* et un *ID_canal* soit un code d'erreur.
- CLOSE *ID_canal* : Cette instruction permet de fermer un canal préalablement ouvert. Elle retourne un code *OK* ou un code d'erreur.
- ENCRYPT *ID_canal, Taille_en-têtes, Taille_données* : Cette instruction est utilisée pour chiffrer un paquet de données appartenant au canal identifié par *ID_canal*. *Taille_en-têtes* et *Taille_données* correspondent respectivement à la taille des données à authentifier seulement et à la taille des données à authentifier et à chiffrer. Avant d'activer le signal *done,* le *contrôleur principal* redirige l'entrée du *crossbar* vers la FIFO du cœur cryptographique qui effectuera le traitement du paquet entrant. Cette instruction renvoie soit un code *OK* et un identifiant de requête *ID_requête,* soit un code d'erreur lorsqu'aucun cœur n'est disponible pour traiter le paquet en attente.
- DECRYPT *ID_Canal, Taille_en-têtes, Taille_données* : Cette instruction fonctionne de façon inverse à l'instruction ENCRYPT dans la mesure où elle permet de déchiffrer un paquet de données. Elle retourne soit un code *OK* et un *ID_requête,* soit un code d'erreur lorsqu'aucun cœur n'est disponible. Comme pour les instructions ENCRYPT, *le contrôleur principal* configure le *crossbar* lorsqu'un code *OK* est retourné.

- RETRIEVE_DATA : Cette instruction est utilisée par le processeur hôte pour récupérer les données une fois qu'un évènement *data_available* a été capturé. Elle retourne soit un code *OK,* soit un code *AUTH_FAIL* lorsque l'authentification d'un paquet a échoué. L'*ID_requête* de l'instruction ENCRYPT/DECRYPT correspondante est aussi retournée de manière à spécifier l'origine des données traitées. Comme pour les instructions ENCRYPT et DECRYPT, *le contrôleur principal* configure le *crossbar* lorsqu'un code *OK* est retourné.
- TRANSFER_DONE : Cette instruction est utilisée par le *processeur hôte* pour notifier au MCCP que l'ensemble d'un paquet a été transféré vers/depuis la FIFO d'entrée/de sortie du cœur cryptographique après l'exécution réussie d'une instruction ENCRYPT, DECRYPT ou RETRIEVE_DATA. Lorsque cette instruction est exécutée par le contrôleur principal celui-ci désactive l'accès en écriture/lecture au cœur cryptographique impliqué dans le transfert de données.

4.2.4 Attribution des tâches de traitement des paquets aux cœurs cryptographiques

Le *crossbar d'entrée* du MCCP n'étant pas équipé de buffer, il n'est donc pas possible pour le MCCP de retarder le traitement d'un paquet faiblement prioritaire. Le MCCP supporte donc exclusivement un algorithme d'ordonnancement du type FIFO pour lequel le premier arrivé est le premier servi. Un tel comportement permet d'optimiser l'utilisation des ressources matérielles et de maximiser le débit de sortie du MCCP, la contrepartie étant un impact négatif sur la latence ou encore la gigue introduite par le MCCP. Le niveau de latence d'un canal de communication étant une caractéristique cruciale pour les protocoles de communications temps réel, l'ordonnancement du type FIFO n'est pas des plus adaptés. Toutefois, le processeur hôte situé en amont du MCCP peut tout à fait ordonnancer les paquets avant de les transmettre au MCCP.

Le *contrôleur principal* du MCCP n'ayant pas pour tâche d'ordonnancer les paquets, il lui reste donc à gérer l'attribution du traitement des paquets aux différents cœurs cryptographiques. L'algorithme actuel *d'attribution des tâches,* exécuté par le *contrôleur principal*, repose sur le comportement suivant : lorsqu'une instruction ENCRYPT ou DECRYPT est reçue par le *contrôleur principal,* celui-ci parcourt l'ensemble des cœurs (toujours dans le même sens) jusqu'à en trouver un inactif qui se verra attribuer la tâche. Si aucun cœur n'est disponible, un code d'erreur est retourné. Afin d'éviter qu'un appel aux fonctions de chiffrement ou déchiffrement ne retourne une erreur, le processeur hôte devrait donc maintenir une représentation de l'état du MCCP. Le principal défaut de cette méthode réside dans le fait que le cœur examiné en premier par le *contrôleur principal* du MCCP est de facto plus utilisé que les autres. Cela peut éventuellement conduire à une détérioration plus rapide d'une partie du FPGA. Il est possible de pallier à ce problème en mettant en place un algorithme de *round-robin* [Gila08] chargé de sélectionner le prochain cœur à utiliser. Ce type d'algorithme permet d'équilibrer la charge d'utilisation de plusieurs ressources.

4.3 Implantation et fonctionnement des cœurs cryptographiques

4.3.1 Implantation des cœurs cryptographiques

L'architecture d'un cœur cryptographique est illustrée en Figure 41. Sur cette figure, les flèches en pointillés correspondent aux signaux de contrôle tandis que les flèches en trait plein correspondent à des bus de données/adresses. Chaque *cœur cryptographique* est constitué de divers composants. Tout d'abord, les services cryptographiques (AES-CTR, CBC-MAC, CCM et GCM) sont fournis par *l'unité cryptographique* dont la gestion est déléguée au *contrôleur 8 bits* embarqué dans le cœur. Ce dernier est programmé de manière à générer un flot d'instructions destiné à être exécuté par *l'unité cryptographique*. Le flot d'instruction généré par le *contrôleur 8 bits* est spécifique au mode de chiffrement mis en œuvre. Il est construit à partir du jeu d'instruction de *l'unité cryptographique*, celui-ci est étudié plus en détail dans la partie suivante. Ensuite, *le cache de clés* permet le stockage des clés de round AES nécessaires au chiffrement et déchiffrement des paquets. Pour finir, les cœurs communiquent avec le processeur hôte et les autres cœurs cryptographiques par l'intermédiaire de quatre ports de données différents : deux ports pour les opérations d'E/S vers le processeur hôte (en haut de la figure) et deux *ports de communication inter-cœur* pour les opérations d'E/S vers les cœurs adjacents (à gauche et à droite de la figure). Le caractère asynchrone des communications entre un cœur cryptographique et le reste du système est rendu possible par l'utilisation de deux FIFO (512 x 32 bits) et d'un registre à décalage (4 x 32 bits).

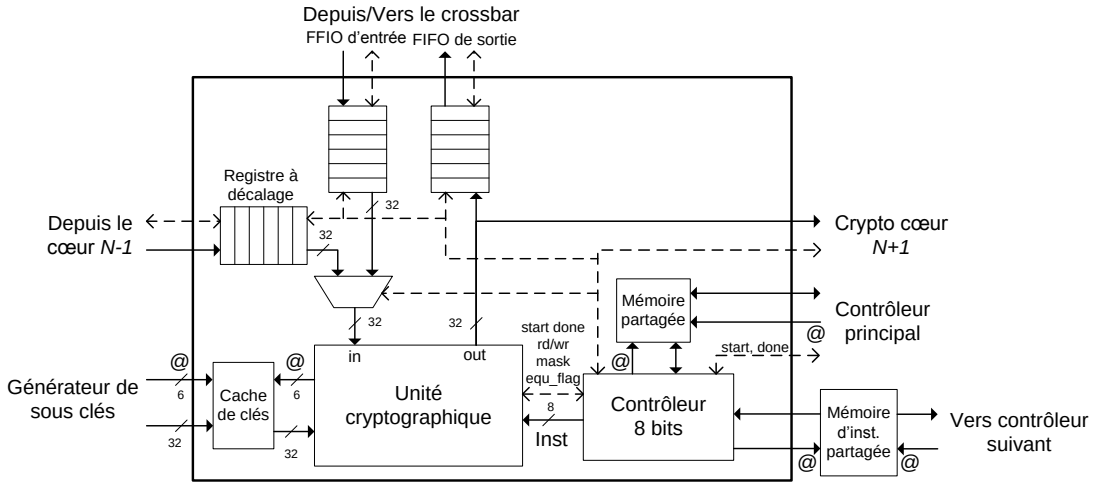


Figure 41 : Architecture d'un cœur cryptographique

Les ports de communication *inter-cœur* rendent possible le transfert de données entre un cœur cryptographique et le cœur cryptographique situé directement à sa droite. Ces ports rendent possible le traitement d'un *même* paquet par *plusieurs* cœurs. Ce mode de fonctionnement est particulièrement utile lors de l'exécution du mode de chiffrement CCM (cf. Annexe C) lorsque chaque *unité cryptographique* n'embarque qu'une seule primitive de

chiffrement AES. En effet, lorsqu'un paquet a besoin d'être chiffré à l'aide du mode de chiffrement CCM, l'exécution du mode CBC-MAC peut être attribuée au premier cœur tandis que l'exécution du mode CTR est attribuée au second. Le *port de communication inter-cœur* est alors utilisé pour le transfert, depuis le cœur CBC-MAC vers le cœur CTR, des données du paquet et de la valeur MAC. Le débit total se trouve alors doublé par rapport au cas où une seule primitive de chiffrement AES est utilisée (mais deux cœurs sont alors utilisés). Il est à noter que les *ports de communication inter-cœur* perdent une partie de leur intérêt lorsque l'on considère la possibilité de reconfigurer dynamiquement *l'unité cryptographique* dans le but d'y embarquer deux primitives de chiffrement. En effet, il est alors possible d'exécuter le mode chiffrement CCM sur un seul cœur cryptographique tout en profitant d'un débit équivalent à celui de la solution basée sur l'utilisation de deux cœurs cryptographiques. Les temps de reconfiguration dynamique pouvant toutefois être pénalisant, nous avons fait le choix de conserver ces ports de communication.

Comme dit précédemment, les *cœurs cryptographiques* doivent être compatibles avec plusieurs modes de chiffrement (CTR, CBC-MAC, CCM et GCM). Or, l'utilisation d'une machine d'état pour le contrôle de ces cœurs mènerait à l'implantation de circuits multimodes excessivement complexes. Afin de réduire la complexité des cœurs tout en améliorant leur interopérabilité, il a été décidé d'implanter leur logique de contrôle sous la forme d'un contrôleur programmable. Ce dernier est chargé de générer un flot d'instructions exécutable par *l'unité cryptographique*. En outre, il est aussi chargé de gérer les communications entre le cœur et le *contrôleur principal* du MCCP. L'utilisation d'un contrôleur programmable à la place d'une machine d'état permet de simplifier la mise en œuvre des boucles et des structures de contrôle (ex. if-else, for, etc.) nécessaires à l'implantation des modes de chiffrement.

Étant donné que ce contrôleur n'effectue aucun calcul complexe, un simple contrôleur 8 bits de type *Picoblaze* [Ken00] a été utilisé pour son implantation. Celui-ci embarque un banc de registre de 16 x 8 registres pour sa version Virtex 4 (et 32 x 8 pour sa version Virtex 6) et quelques opérateurs logiques et mathématiques suffisants pour notre application. Chaque instruction est exécutée en deux cycles d'horloge et le nombre d'instructions est limité, pour la version V4, à 1024 instructions stockées dans un bloc RAM du FPGA (4096 pour la version V6). Le contrôleur *Picoblaze* supporte aussi la gestion des interruptions. Pour finir, une instruction HALT permettant d'interrompre le fonctionnement du contrôleur jusqu'à la levée d'un signal d'interruption a été ajoutée au Picoblaze dans sa version V4 (dans sa version V6 le signal de mise en veille *sleep* présent de façon standard en entrée du contrôleur est utilisé). Par la suite (cf. partie 3.5), nous reviendrons plus longuement sur l'intérêt que peut avoir la mise en pause du fonctionnement du contrôleur embarqué dans les cœurs cryptographiques.

L'architecture des cœurs cryptographiques ayant été décrite, le processus de traitement des paquets et la mise en œuvre des modes de chiffrement peuvent maintenant être détaillés.

4.3.2 **Traitement des paquets et mise en œuvre des modes de chiffrement**

Une fois le traitement d'un paquet attribué à un cœur cryptographique, le processus de traitement est effectué de la manière suivante :

1. D'abord, le *contrôleur principal* lance si besoin est la génération des sous clés cryptographiques. Ensuite, il envoie au contrôleur du cœur cryptographique sélectionné les paramètres de configuration nécessaires au traitement du paquet de données. Pour finir, il envoie un signal *start* au cœur cryptographique.
2. Le contrôleur du cœur cryptographique commence alors les calculs préalables nécessaires à l'initialisation du mode de chiffrement sélectionné.
3. Une fois le mode de chiffrement initialisé, les données sont lues par bloc de 128 bits depuis la *FIFO d'entrée* avant d'être traitées par le cœur. Le résultat du traitement d'un bloc est écrit soit dans la *FIFO de sortie* soit dans le registre à décalage du *port de communication inter-cœur*.
4. Une fois que tous les blocs de données ont été traités, le contrôleur du cœur notifie, via le signal *done,* la fin du traitement au *contrôleur principal* du MCCP.

Afin de protéger le processeur hôte d'éventuelles attaques logicielles, la FIFO de sortie est réinitialisée lorsque l'authentification d'un paquet échoue. De cette façon, le processeur hôte n'est jamais exposé à un paquet corrompu.

Les étapes 2 et 3 varient en fonction du mode de chiffrement mis en œuvre et de la configuration des cœurs. Toutefois, le traitement d'un paquet suit toujours les *modes de traitement* suivants :

- Les paquets en provenance d'un même canal peuvent être traités en parallèle sur différents cœurs.
- Les paquets en provenance de différents canaux peuvent être traités en parallèle sur différents cœurs.
- N'importe quel paquet peut être traité par n'importe quel cœur. Toutefois, il se peut qu'il soit au préalable nécessaire de reconfigurer dynamiquement l'*unité cryptographique*. Lorsqu'une telle reconfiguration n'est pas envisageable, l'*unité cryptographique* doit être configurée de façon à être compatible avec l'ensemble des modes (c'est-à-dire : AES et multiplieur de Galois). En conclusion, bien qu'étant intrinsèquement homogène, le MCCP peut être modifié de manière à se comporter comme un cryptoprocesseur multicœur homogène ou hétérogène.
- Le traitement d'un paquet nécessitant un chiffrement CCM peut être distribué sur deux cœurs cryptographiques. Lorsqu'il est possible de reconfigurer dynamiquement les cœurs cryptographiques, l'intérêt de cette pratique diminue.

Le Tableau 14 page 102 récapitule l'ensemble des différents modes de fonctionnement possible et précisent leurs performances. Le *mode traitement* à appliquer à un paquet est

sélectionné par le *contrôleur principal* du MCCP lors de l'étape d'attribution du traitement d'un paquet à un cœur cryptographique.

4.4 L'unité cryptographique

4.4.1 Architecture de l'unité cryptographique

L'unité cryptographique fournit un support bas niveau des primitives cryptographiques. L'architecture de *l'unité cryptographique*, illustrée en Figure 42, est conçue de manière à être compatible avec un large nombre de modes de chiffrement (CTR, CBC-MAC, CCM et GCM). Elle embarque pour cela un bloc de chiffrement AES, un cœur de hachage GHASH ainsi que quelques opérateurs logiques et arithmétiques (XOR, additionneur, comparateur). Dans sa version dynamiquement reconfigurable, le cœur de hachage GHAS peut être reconfiguré et remplacé par un cœur de chiffrement AES. L'*unité cryptographique* embarque aussi un décodeur d'instructions et un registre de taille 16 x 32 bits divisé en quatre pages, chaque page peut servir au stockage d'un bloc de données codé sur 128 bits. L'adressage des pages du registre se fait par l'intermédiaire des champs d'adresse contenus dans les instructions. L'adressage des mots composant les pages est effectué grâce à un compteur modulo quatre contrôlé par le décodeur. Le chemin de donnéess de *l'unité cryptographique* est codé sur 32 bits, quatre cycles d'horloge sont donc nécessaires à l'accès d'un bloc de 128 bits de données. L'unité cryptographique accède aux quatre mots de 32 bits de façon séquentielle. Pour finir, un registre *start* est utilisé pour déclencher l'exécution d'une instruction.

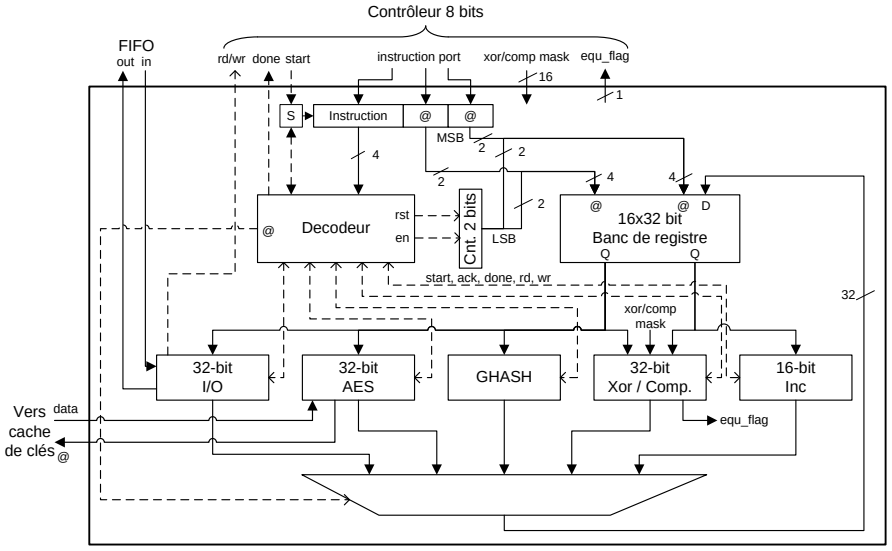


Figure 42 : Architecture de l'unité cryptographique

L'architecture du cœur de chiffrement AES implanté est basée sur les travaux de P. Chodowiec et K. Gaj [ChGa03] qui portent sur l'implantation sur FPGA d'un cœur de

chiffrement AES compact. Les modes CCM et GCM n'utilisent pas le déchiffrement AES, celui-ci n'a donc pas été implanté. Le cœur de chiffrement AES est implanté sous la forme d'une architecture itérative et la transformation SuBytes d'AES [Nist01a] est réalisée grâce à des tables de correspondance stockées dans des blocs RAM. L'architecture étant itérative, le nombre de cycles nécessaires au chiffrement d'un bloc de données est donc fonction de la taille de la clé de chiffrement. Le chiffrement d'un bloc de données de 128 bits nécessite 44, 52 ou 60 cycles selon que la clé de chiffrement utilisée fait 128, 192 ou 256 bits.

Le cœur GHASH est basé sur l'architecture de multiplieur série (*digit-serial*) décrite dans [LWFB07]. La multiplication chiffre-série est effectuée à partir de chiffres codés sur trois bits, elle est donc effectuée en 43 cycles d'horloge.

Le cœur XOR/Comparateur prend deux mots de 128 bits en entrée et renvoie la valeur $B = (A \oplus B) \cdot mask$ ou met le signal *equ_flag* à 1 si A et B sont égaux. Dans le premier cas, le champ *mask* permet à l'utilisateur de masquer des octets spécifiques du résultat fourni par l'opérateur XOR. Ce genre de masquage est utile lorsque le mot binaire à chiffrer avec le mode CTR est d'une longueur inférieure à 128 bits. L'opérateur d'incrémentation permet l'incrémentation par pas de un à quatre des 16 bits de poids faibles d'un mot de 128 bits. L'incrémentation est limitée au 16 bits de poids faibles afin de réduire le chemin critique de l'additionneur, cela n'a pas de conséquence sur les opérations de chiffrement dans la mesure où seulement 128 incrémentations du vecteur d'initialisation sont nécessaires au chiffrement d'un paquet de 2048 Ko avec les modes CTR, CCM et GCM. Pour finir, le module d'E/S permet le transfert de blocs de données entre le banc de registre interne à *l'unité cryptographique* et les FIFO d'entrée et de sortie du *cœur cryptographique*.

Tableau 15 : Jeu d'instructions de l'unité cryptographique

Instruction	Opérandes	Description
LOAD	@A	Charge un mot de 128 bits depuis la FIFO d'entrée ou le cœur précédent dans le banc de registres.
STORE	@A	Envoie le contenu du registre A dans la FIFO de sortie ou sur le *port de communication inter-cœur*.
LOADH	@A	Charge la constante de multiplication H dans le multiplieur de Galois.
SGFM (non bloquant)	@A	Effectue une itération de l'algorithme GHASH.
FGFM	@A	Sauvegarde le résultat de l'algorithme GHASH dans le registre A.
SAES (non bloquant)	@A	Chiffre la valeur stockée dans le registre A.
FAES	@A	Sauvegarde le résultat de la dernière exécution de SAES dans le registre A.
INC	@A, I	Incrémente par I les 16 bits de poids faible du registre A, $I \in \{1, 2, 3, 4\}$.
XOR	@A, @B	Calcule $B = (A \oplus B) \cdot mask$.
EQU	@A, @B	Met le signal *equ_flag* à 1 si A = B et 0 sinon.

4.4.2 **Jeu d'instructions et fonctionnement de l'unité cryptographique**

L'*unité cryptographique* est contrôlée grâce au jeu d'instructions détaillé dans le Tableau 15. Celui-ci a été conçu dans le but de répondre aux besoins des *cœurs cryptographiques*. Il fournit des instructions d'E/S et des instructions cryptographiques de bas niveau. Les instructions sont codées sur 8 bits et se décomposent en trois champs. Le premier consiste à un code d'instruction codé sur quatre bits et les deux autres correspondent aux adresses des opérandes qui sont codés sur deux bits. L'exécution d'une instruction par l'*unité cryptographique* est faite en six cycles d'horloge depuis le front montant du signal *start* jusqu'au front montant du signal *done*. Toutefois, les instructions SAES et SGFM ont un temps d'exécution supérieur à six cycles. Ces instructions sont donc conçues de manière à permettre à l'exécution concurrente d'autres instructions en fonctionnant en tâche de fond.

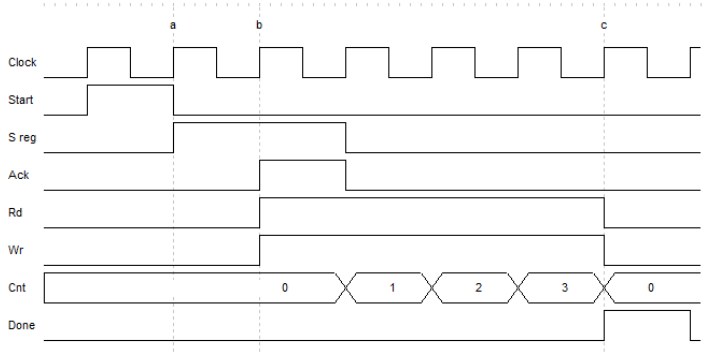


Figure 43 : Chronogramme d'exécution d'une instruction XOR

À titre d'exemple, l'exécution d'une instruction XOR, illustrée Figure 43, suit les étapes suivantes :

1. Lorsque le signal *start* passe à l'état haut, la valeur présente sur le port d'instructions est stockée dans le *registre d'instruction* en même temps que le registre *S* est mis à 1 (Figure 43 marqueur a).
2. Le *décodeur* décode immédiatement l'instruction stockée dans le *registre d'instruction*. Il redirige alors la sortie du registre *S* vers l'entrée de démarrage du module correspondant à l'instruction qui est dans cet exemple *xor_start*. Il redirige aussi la sortie du module XOR vers l'entrée du banc de registres et donne les droits d'accès en lecture et écriture au module.
3. La machine d'état embarquée dans le module XOR envoie un signal *Ack* qui indique par la remise à zéro du registre *S* que le signal *start* a bien été pris en compte (Figure 43 marqueur b).
4. Puis les signaux *rd* et/ou *wr* sont mis à l'état haut dans le but de lire et/ou écrire des données dans le banc de registres. La mise à l'état haut des signaux *rd* et *wr* déclenche automatiquement l'incrémentation du compteur 2 bits qui permet de

parcourir séquentiellement les quatre mots de 32 bits composants les opérandes codés sur 128. La sélection des opérandes se fait par l'intermédiaire des champs d'adresse du registre d'instruction.

Comme cela a été expliqué, le fonctionnement des instructions SAES, SGFM diffère quelque peu du fonctionnement des autres instructions. Un exemple d'utilisation des instructions SAES et FAES est illustré Figure 44. Les instructions SAES et SGFM sont utilisées pour charger les données à traiter dans les cœurs de calcul et lancer les calculs AES et GHASH (Figure 44 marqueur a). Les six premiers cycles des instructions SAES et SGFM (car accès au banc de registre) sont bloquants, les cycles d'exécution restants (calculs AES et GHASH) sont effectués en tâche de fond. Ces six premiers cycles étant passés, d'autres instructions peuvent alors être exécutées (Figure 44 marqueur b). Lorsqu'il est nécessaire d'utiliser le résultat d'un calcul AES ou GHASH, les instructions FAES et FGFM sont utilisées. Elles permettent d'abord d'attendre la fin des calculs AES et GHASH éventuellement en cours, puis de stocker le résultat obtenu dans le banc de registres (Figure 44 marqueur c).

Le mécanisme de synchronisation permettant le fonctionnement des instructions FGFM et FAES est basé sur l'utilisation des signaux *Ack* et du registre *S*. Par exemple, lorsqu'une instruction FAES doit être exécutée, le registre *S* est mis à 1 puis reste à 1 jusqu'à ce que le cœur AES redevienne inactif (Figure 44 marqueur d). À ce moment, la valeur du registre *S* est prise en compte et le signal *Ack* le réinitialise à zéro (Figure 44 marqueur e). De cette façon, les instructions FAES et FGFM sont mémorisées jusqu'à la fin des calculs AES et/ou GHASH.

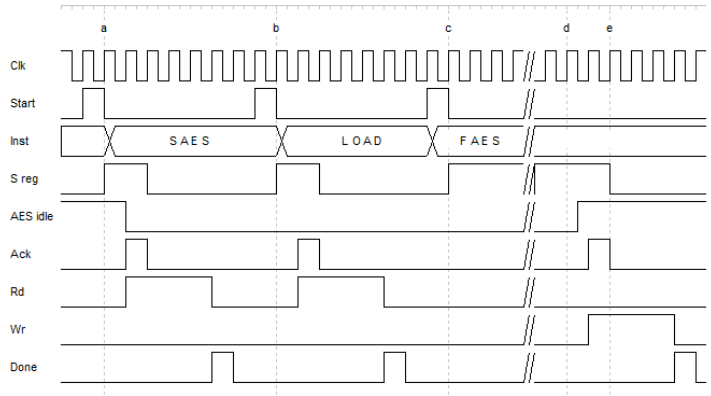


Figure 44 : Chronogramme d'exécution des instructions SAES et FAES

Le jeu d'instruction de *l'unité cryptographique* ayant été présenté, il est maintenant possible de décrire l'implantation logicielle des modes de chiffrement.

4.4.3 **Implantation logicielle des algorithmes cryptographiques**

Avant de pouvoir implanter de façon logicielle les algorithmes cryptographiques impliqués dans le chiffrement et le déchiffrement des paquets de données, il faut décrire la façon dont un paquet est traité. Tout d'abord, le *contrôleur principal* du MCCP sélectionne le cœur qui est chargé du traitement d'un paquet. Par la suite, il lance le processus de génération des sous clés cryptographiques et transmet la configuration du canal de chiffrement et les caractéristiques du paquet au *cœur cryptographique* (notamment l'algorithme de chiffrement, les tailles des champs d'en-tête et de corps de message et la taille du tag d'authentification). Enfin, le contrôleur principal émet un signal *start* à l'attention du *cœur cryptographique* et autorise la transmission des données depuis le processeur hôte vers le *cœur cryptographique*. À ce stade la transmission du paquet au cœur peut commencer. Il est à noter que les données doivent être envoyées d'une manière spécifique : d'abord le vecteur d'initialisation doit être chargé dans la FIFO d'entrée ensuite, c'est au tour du corps du paquet. Pour finir, dans le cas d'un déchiffrement, le tag d'authentification doit être chargé. Étant donné que *l'unité cryptographique* implantée dans ce prototype n'embarque qu'un petit nombre d'opérateurs arithmétiques et logiques simples, elle ne peut pas être utilisée pour formater les paquets selon les spécifications du NIST (notamment en ce qui concerne les en-têtes des paquets CCM [WhHF02]). Par conséquent, le processeur hôte doit au préalable formater les paquets de données transmis au MCCP. Une fois le paquet correctement formaté et en cours de transmission au *cœur cryptographique*, le traitement des données peut commencer.

Nous avons vu que le flot d'instructions exécuté par *l'unité cryptographique* et servant à traiter les paquets était généré par le contrôleur *Picoblaze* du *cœur cryptographique*. La génération d'une instruction de ce flot suit trois étapes : d'abord, l'instruction à exécuter est chargée dans le banc de registre du contrôleur ensuite, l'instruction est écrite sur le port d'instructions de *l'unité cryptographique* et enfin, le contrôleur attend que l'exécution de l'instruction se termine. Afin d'améliorer les performances, il est possible de précharger dans le banc de registres les instructions qui seront exécutées.

Le chargement et le pré-chargement des instructions sont effectués avec l'instruction assembleur LOAD du Picoblaze qui permet le chargement immédiat dans le banc de registres d'un octet de données. L'étape d'exécution est mise en œuvre grâce à l'instruction OUTPUT du Picoblaze qui recopie un octet de données depuis le banc de registres vers le port de sortie du Picoblaze. De plus, le signal d'écriture du port de sortie est directement connecté au port *start* de *l'unité cryptographique* de manière à lancer immédiatement l'exécution de l'instruction. Pour finir, une instruction HALT (pour la version Virtex 4) ou un registre HALT (pour la version Virtex 6) est utilisé pour mettre en pause le fonctionnement du contrôleur jusqu'à la fin de l'exécution de l'instruction envoyée à *l'unité cryptographique*. On peut remarquer que les temps d'exécution des instructions autres que FAES ou FGFM sont connus (six cycles), une instruction HALT peut donc être remplacée par deux instructions NOP. De cette façon, le contrôleur n'attend pas la mise à 1 prévisible du signal *done* pour sortir de son état inactif. Cette technique d'optimisation permet d'économiser un cycle d'horloge par instruction exécutée.

```
GCMloop :       OUTPUT   FAES, inst
                HALT     DISABLE
                OUTPUT   SAES, inst
                OR       s0, FF             ;NOP
                OR       s0, FF             ;NOP
                OUTPUT   IXOR, inst
                OR       s0, FF             ;NOP
                OR       s0, FF             ;NOP
                OUTPUT   SGFM, inst
                HALT     DISABLE
                OUTPUT   STORE, inst
                OR       s0, FF             ;NOP
                OR       s0, FF             ;NOP
                OUTPUT   INC, inst
                OR       s0, FF             ;NOP
                OR       s0, FF             ;NOP
                OUTPUT   LOAD_PT, inst
                SUB      dlength, 01
                JUMP     NZ, GCMloop
```

Listing 1 : Corps de la boucle d'exécution du mode GCM

La partie critique de l'implantation d'un algorithme cryptographique réside dans la boucle principale de l'algorithme qui effectue le chiffrement de chacun des blocs de données constitutif d'un paquet. Le fait qu'un paquet puisse être constitué de 128 blocs de données démontre clairement cette criticité. Le Listing 1 présente la partie du code de l'algorithme GCM qui correspond au corps de sa boucle principale. Ce listing illustre la façon dont le code assembleur est optimisé en montrant par exemple que plusieurs instructions sont insérées entre les instructions SAES et FAES. Il montre aussi que lorsque le temps d'exécution d'une instruction peut être estimé, des NOP sont utilisés à la place de l'instruction HALT.

Tableau 16 : Temps de calcul pour le chiffrement d'un bloc de données

Mode de chiffrement	Temps de chiffrement d'un bloc de données		
	En modes de chiffrement simples	En opérations	En cycles
GCM	T_{CTR}	T_{AES}	49
CCM (2 cœurs)	T_{CBC}	$T_{AES} + T_{XOR}$	55
CCM (1 cœur)	$T_{CTR} + T_{CBC}$	$2*T_{AES} + T_{XOR}$	104

Au niveau le plus bas, les performances globales sont limitées par l'accélérateur AES qui possède le chemin critique le plus important [ChGa03]. À plus haut niveau, ce sont les boucles principales des modes de chiffrement qui limitent le débit global du MCCP. Le temps de calcul de ces boucles peut être utilisé pour approximer les performances des modes de chiffrement implémentés. Le Tableau 16 présente les temps de calcul nécessaires au chiffrement d'un bloc de données (ce qui est équivalent à un tour de boucle) avec une clé de 128 bits en fonction du mode de chiffrement utilisé. Dans la mesure où le nombre de rounds AES varie en fonction de la taille de clé, huit cycles doivent être ajoutés aux valeurs fournies par le tableau pour une clé de 192 bits et seize cycles doivent être ajoutés pour une clé de 256 bits.

Afin d'estimer le débit maximal du MCCP, il faut prendre en compte les temps d'exécutions des instructions qui précèdent et suivent la boucle principale des modes de chiffrement. Ces temps d'exécution, qui sont invariants, ont été mesurés par simulation. Ils sont de 337 cycles pour AES-GCM, de 354 cycles pour AES-CCM (2 cœurs) et de 710 cycles pour AES-CCM (1 cœur).

Pour finir, les temps d'accès aux cœurs cryptographiques doivent être pris en compte. Les délais en écriture peuvent être négligés dans la mesure où le *cœur cryptographique* commence son travail à partir du moment où au moins un bloc complet de données est disponible dans la FIFO d'entrée. Au contraire, les délais en lecture ne peuvent pas être négligés, car le *cœur cryptographique* reste inactif pendant tout le processus de lecture.

En résumé, le temps de calcul total nécessaire au traitement d'un paquet peut être approximé par la formule (1) où :

- Le premier terme correspond à la portion du temps de traitement dont la durée dépend de la taille du paquet.
- Le second terme correspond à la portion du temps de traitement dont la durée est invariable. Cette durée dépend seulement du mode de chiffrement exécuté.
- Le troisième terme correspond au délai de transmission des données depuis la FIFO de sortie du cœur cryptographique vers le processeur hôte.

$$T_{total} = T_{boucle} * taille_{paquet} + T_{inv} + taille_{paquet}/taille_{bus} \quad (1)$$

4.5 Résultats d'implantation et études des performances

Cette partie présente les résultats obtenus suite à l'implantation de l'architecture du MCCP présentée dans ce chapitre.

4.5.1 Ressources matérielles occupées

L'architecture du MCCP présentée dans ce chapitre a été décrite en VHDL puis synthétisée grâce à la suite logicielle Xilinx ISE 12.1. L'implantation matérielle a été faite sur deux FPGA Xilinx différents : un Virtex 4 SX35-11 et un Virtex 6 LX240T-1. Nous verrons dans la suite de cette partie que les performances (ex. nombre de slices ou encore fréquence maximale de fonctionnement) obtenues varient en fonction du type de FPGA utilisé pour l'implantation matérielle. Ces variations ont plusieurs origines et notamment le fait que les puces Virtex 4 et Virtex 6 ne sont pas basées sur les mêmes structures élémentaires. En outre ces deux puces ne sont pas construites avec la même technologie et donc la même densité d'intégration, le Virtex 4 est un circuit en technologie SRAM à 90nm alors que le Virtex 6 est un circuit en technologie SRAM à 40nm. Le Tableau 17 récapitule l'ensemble des caractéristiques des FPGA Virtex 4 et Virtex 6 utilisés au cours de ces travaux de recherche.

Tableau 17 : Caractéristique des FPGA Virtex 4 et 6 utilisés pour ces travaux

Caractéristique	Virtex 4 SX35-11 - 90nm	Virtex 6 LX240T-1 - 40nm
Nb d'entrées des LUT	4	6
Nb LUT /FF par slice	2 / 2	4 / 8
Taille des blocs RAM (Kb)	18	36
Nb slices	15 360	37 680
Nb BRAM	192	416

Les résultats d'implantation du MCCP, suite à la synthèse et au placement et routage, sont détaillés dans le Tableau 18. Ce tableau donne le nombre de slices, de BRAM et la fréquence maximale de fonctionnement du MCCP avec un circuit Virtex 4 (noté *V4* dans le Tableau 18) et avec un circuit Virtex 6 (noté *V6* dans le Tableau 18) en fonction du nombre de cœurs embarqués. Le nombre de BRAM différent entre les implantations sur Virtex 4 et Virtex 6 vient du fait que chaque *cœur cryptographique* de la version Virtex 6 du MCCP possède sa propre mémoire d'instruction alors que celle-ci est partagée entre deux cœurs dans la version Virtex 4. Pour le reste, les deux circuits sont fonctionnellement semblables.

Tableau 18 : Résultats d'implantation du MCCP sur Virtex 4 et 6

Nb de cœurs cryptographiques	Taille de bus	Slices		BRAM		Fréquence Max.	
		V4	V6	V4	V6	V4	V6
2	32	2071	841	15	16	192	181
4	32	4055	1710	26	28	192	170
6	32	6026	2501	37	40	185	167
8	32	7760	3189	48	52	192	165
8	64	8714	3157	64	68	192	162

Concernant le nombre de slices occupées, on remarque, pour les deux circuits visés, que celui-ci augmente linéairement avec le nombre de *cœurs cryptographiques* implantés. Le circuit en version 32 bits utilise 2,4 fois plus de slices pour un circuit Virtex 4 que pour un circuit Virtex 6. Cette différence s'explique par le fait qu'une slice d'un circuit Virtex 6 embarque deux fois plus de LUT qu'une slice d'un circuit Virtex 4.

La différence est plus importante pour la version du MCCP qui embarque huit cœurs cryptographiques. En effet, sur Virtex 4, la version 64 bits du circuit nécessite 1000 slices de plus que sa version 32 bits. Au contraire, le nombre de slices occupées est quasi identique sur Virtex 6. Cette différence est due au nombre d'entrées supérieur des LUT du Virtex 6 qui permettent une implantation plus compacte (en nombre de LUT) des multiplexeurs. Pour finir, on peut remarquer que notre MCCP occupe 56% d'un FPGA Virtex 4 SX35 contre seulement 8,4% sur Virtex 6 LX240T. La diminution de la finesse de gravure des FPGA (de 90 nm pour les Virtex 4 à 28 nm pour les Virtex 7) laisse entendre qu'à l'avenir une part toujours plus grande d'une radio logicielle pourra être embarquée sur un unique FPGA.

L'étude de la fréquence de fonctionnement maximale du MCCP avec les circuits Virtex 4 et Virtex 6 expose des profils radicalement différents. Alors que la fréquence de fonctionnent du MCCP implanté dans un circuit Virtex 4 reste constante quel que soit le nombre de cœurs cryptographiques (à l'exception de la version à six cœurs du MCCP dont l'optimisation semble poser problème à l'outil de synthèse XST de Xilinx), cette fréquence diminue avec l'augmentation du nombre de cœurs sur Virtex 6. Le chemin critique du MCCP, identique quel que soit le composant choisi, se situe entre le port de sortie des Picoblaze embarqués dans les *cœurs cryptographiques* et leur mémoire d'instructions. La raison de cette différence de comportement n'a pas pu être précisément déterminée. Toutefois, on peut avancer les hypothèses suivantes : les outils ISE pour Virtex 6 n'ont pas encore atteint une maturité suffisante par rapport à leur équivalent pour Virtex 4 ou le Picoblaze n'est pas aussi bien optimisé pour les circuits Virtex 6 que pour les circuits Virtex 4. Une solution efficace à ce problème serait de pipeliner les contrôleurs intégrés aux *cœurs cryptographiques*, une autre solution plus incertaine serait d'implanter les SBOX du bloc AES sous forme de LUT afin de libérer quelques blocs RAM et simplifier le travail de l'outil de placement.

Tableau 19 : Débit d'un cœur cryptographique à 190 MHz (Débit limite / 2 Ko)

Taille de clé (bits)	AES-GCM (Mbps)		AES-CCM (Mbps)			
	1 cœur par paquet		1 cœur par paquet		2 cœurs par paquet	
	Débit limite	Paquet de 2 Ko	Débit limite	Paquet de 2 Ko	Débit limite	Paquet de 2 Ko
128	496	437	233	214	442	393
192	426	382	202	187	386	348
256	374	337	178	171	342	313

4.5.2 **Performances des cœurs cryptographiques**

Cette section détaille les performances brutes des *cœurs cryptographiques* composant le MCCP. Elles sont tirées de l'implantation du MCCP sur Virtex 4. Le Tableau 19 résume le débit en sortie d'un *cœur cryptographique* pour différentes configurations du cœur, différentes tailles de clés et différentes tailles de paquets. Ce tableau montre que dans le cas d'AES-CCM, il vaut mieux traiter deux paquets différents sur deux cœurs différents qu'un même paquet sur des cœurs fonctionnant en parallèle. Autrement dit, le traitement d'un paquet sur un seul cœur est plus efficace que le traitement du même paquet sur des cœurs fonctionnant en parallèle. Cependant, la latence de traitement d'un paquet avec la solution n'utilisant qu'un cœur par paquet est environ deux fois plus grande que celle utilisant deux cœurs par paquet. Par conséquent, le concepteur devra programmer le contrôleur du MCCP de manière à ce que celui-ci utilise l'option la plus adaptée à l'application ciblée. Le Tableau 19 montre aussi que le débit réel des *cœurs cryptographiques* dépend de la taille des paquets, car un certain nombre d'instructions doivent toujours être exécutées lors d'une opération de chiffrement/déchiffrement et ce quel que soit la taille du paquet traité. Comme le laisse supposer l'équation (1), les meilleurs débits sont obtenus avec les paquets les plus grands. Or, les applications de communications temps réel échangent de nombreux paquets de faible taille : les algorithmes de compression de la voix G.711[Itu00a] et G.729A [Itu00b]

génèrent des paquets dont les tailles sont comprises entre 30 et 240 octets [SzHa04] (les en-têtes IP ou autres ne sont pas comptés). Il s'ensuit que le débit réel est inférieur au débit maximal qui pourrait être fourni par le MCCP

La Figure 45 illustre la variation du débit (en Mbps) en fonction de la taille des paquets (en octets) pour une taille de clé de 128 bits. Cette figure montre que le débit d'un cœur tombe à seulement 50 Mbps pour des paquets de petite taille.

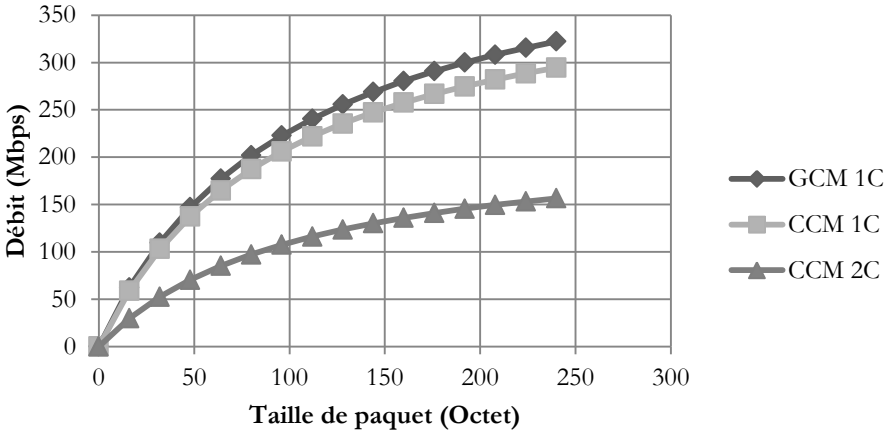


Figure 45 : Débit en sortie d'un cœur cryptographique en fonction de la taille du paquet traité (Mbps/Octet) en fonction du mode de fonctionnement du chiffrement (GCM ou CCM) et du nombre de cœurs cryptographiques utilisés (1 ou 2).

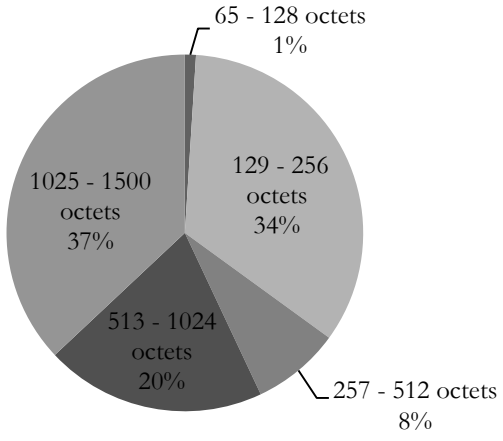


Figure 46 : Distribution des paquets de taille donnée dans un flux vidéo MPEG [SzHa04]

Il faut cependant relativiser cette chute de débit dans la mesure où la bande passante nécessaire au bon fonctionnement des algorithmes G.711 et G.729 est comprise entre 19 Kpbs (G.729) et 80 Kbps (G.711) (les en-têtes IP ou autres ne sont pas comptés) [SzHa04].

Par exemple, une bande passante de 50 Mbps permet *théoriquement* d'assurer le transport d'environ 2600 canaux de communication en mode simplex. En ce qui concerne la vidéoconférence, le problème est plus complexe. La Figure 46 illustre la distribution des paquets de taille donnée dans un flux vidéo MPEG. Elle montre que la taille des paquets vidéo varie au cours du temps. En effet, les images de référence d'un flux MPEG sont faiblement compressées contrairement aux autres images du flux. Le débit en sortie des cœurs cryptographiques est donc amené à varier au cours du traitement d'un flux vidéo. Il est à noter que le débit moyen d'un flux de vidéoconférence de bonne qualité est de 384 Kbps et donc que là encore le MCCP a un débit suffisant pour assurer *théoriquement* le transport de plusieurs centaines de flux vidéo. En réalité, les problématiques liées à la congestion des canaux de communication font que les capacités *réelles* du MCCP sont inférieures aux capacités *théoriques* qui précèdent. Il nous faudrait être en possession d'un prototype complet, chose que nous n'avons pas, pour procéder à des mesures précises.

Jusqu'ici le phénomène de latence induit par le traitement des paquets n'a pas été pris en compte. Il s'avère qu'en ce qui concerne les applications de communications temps réel, les contraintes de latence sont souvent plus difficiles à satisfaire que les contraintes de débit[21]. De plus, une variation du débit en sortie des cœurs implique mécaniquement une variation de la latence de traitement des paquets. Afin d'étudier cette problématique, la partie suivante aborde plus longuement l'étude du débit et de la latence du processeur MCCP.

4.5.3 **Performances globales du MCCP**

L'architecture du MCCP implantée ne permet pas de retarder le traitement d'un paquet afin d'en avantager un autre : les paquets sont traités dans leur ordre d'arrivée. Dans ce cas, c'est au processeur hôte de gérer lui-même le processus d'ordonnancement des paquets. Dans un premier temps, afin de rester aussi général que possible cette partie présente une analyse théorique des performances du MCCP dans le cadre de flux de paquets non ordonnancés. Elle est suivie par une étude pratique issue de simulations. Dans un second temps, nous abordons de façon succincte le cas où les paquets sont ordonnancés avant d'être transmis au MCCP.

Dans le but d'évaluer les performances du MCCP implanté, il est utile de proposer une modélisation des canaux de communication dont le MCCP assure le chiffrement. Généralement, les paquets transitent au travers des canaux de communications de façon apériodique. Toutefois, il existe un délai minimum entre les dates d'arrivée de deux paquets en provenance d'un même canal de communication. Ce délai est au minimum égal à la durée de propagation d'un paquet de données sur le canal de communication. Il s'ensuit que le traitement des paquets appartenant à un canal C_i peut être modélisé par une tâche sporadique $\tau_i = (e_i, p_i)$ grâce au *modèle sporadique de tâche* [Mok83] où :

- e_i est le temps de calcul nécessaire au traitement d'un paquet du canal C_i.

[21] Surtout si en amont les services à l'origine les flux de paquets respectent les débits moyens et les débits crêtes préalablement définis.

- p_i est le délai minimum entre deux arrivées de paquet sur le canal C_i.

Sans perte de généralité, nous considérerons par la suite le cas où les tâches sporadiques permettant de modéliser les canaux de communication se comportent comme des tâches périodiques. Sous cette condition, un ensemble de tâches τ peut être alloué au MCCP si et seulement si la charge réseau n'excède pas les capacités du MCCP ce qui se traduit mathématiquement par l'équation (2) lorsque le MCCP embarque *m* cœurs et que l'on y alloue *n* canaux de communication.

$$U = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq m \quad (2)$$

Cette équation permet d'obtenir théoriquement le débit maximum par canal pour un nombre de canaux de communication donnés. Pour cela, posons :

- p_{min_i} : le délai minimum absolu entre l'arrivée de deux paquets du canal C_i.
- p_i : le délai minimum effectif entre l'arrivée de deux paquets du canal C_i où :

$$p_i = \alpha * p_{min_i} \text{ avec } \alpha \geq 1 \ (3)$$

- e_i : le temps de calcul nécessaire au traitement d'un paquet du canal C_i (ce temps de calcul peut être une majoration lorsque celui-ci n'est pas constant)
- m le nombre de cœurs cryptographiques et n le nombre de canaux.

Le débit maximum théorique est obtenu quand :

$$\frac{1}{m}\sum_{i=1}^{n} \frac{e_i}{p_i} = 1 \Rightarrow \alpha = \frac{1}{m}\sum_{i=1}^{n} \frac{e_i}{p_{min_i}}$$

Si pour simplifier, nous posons que $p_{min_i} = e_i$, nous obtenons alors

$$\alpha = \frac{n}{m} \ (4)$$

Ainsi, un ensemble de canaux est alloué sur le MCCP si et seulement si le délai minimum p_i respecte l'équation (3). Afin de vérifier l'adéquation des performances théoriques avec les performances réelles, un modèle TLM (*Transaction Level Model)* du MCCP a été développé. Dans le but de fournir une estimation précise des performances du MCCP, ce modèle de haut niveau prend en compte les délais de traitement des paquets ainsi que les délais d'accès aux E/S. Le paramétrage de la simulation a été fait à partir des hypothèses suivantes :

- Les paquets ne sont pas ordonnancés : le premier arrivé est le premier servi.

- Les canaux sont chiffrés avec les algorithmes AES-GCM et AES-CCM avec une clé de 128 bits.
- Le débit maximum atteignable par un canal est donné par l'égalité $p_{min_i} = e_i$.

Pour chaque nombre de canaux ouverts, toutes les combinaisons de configuration des canaux ont été simulées et le pire cas a été isolé afin d'obtenir le débit maximum supportable par le MCCP dans le pire des cas. La Figure 47 illustre le graphique obtenu à partir des résultats de simulation. En ordonnée, on retrouve la valeur alpha telle que définie dans l'équation (3) qui a été calculée empiriquement à partir des résultats de simulation. Ce graphique montre que le paramètre alpha suit bien l'équation (4) pour les versions du MCCP embarquant 2, 4 et 6 *cœurs cryptographiques*. Toutefois, à partir de 8 cœurs (courbe 8c32), le débit maximum atteignable est inférieur aux prévisions et seule une augmentation de la taille du bus de communication de 32 à 64 bits permet de retomber sur la formule théorique. Lorsque le nombre de cœurs devient trop important, le bus de communication du MCCP devient un goulot d'étranglement pour le système. Les solutions consistent alors à augmenter la taille du bus ou à augmenter le nombre de processeurs hôtes. Une autre solution consiste à utiliser une architecture de type *processeur réseau*. Ces solutions ont déjà fait l'objet de discussions au chapitre précédent.

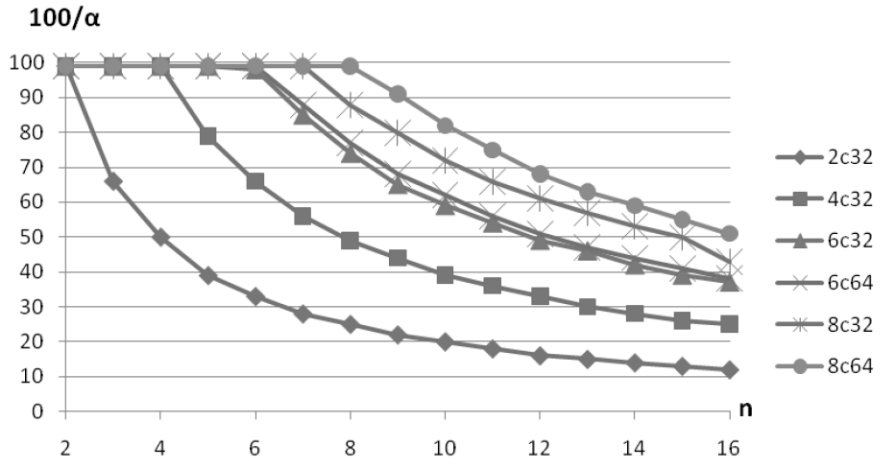


Figure 47 : Variation du délai minimum entre deux arrivées de paquets en fonction du nombre de canaux ouverts

Nous venons de voir que le débit maximum par canal diminue avec l'augmentation du nombre de canaux. Toutefois, l'approche envisagée jusqu'ici est une approche « *best effort* » dans la mesure où nous avons essayé d'optimiser l'utilisation des cœurs du MCCP sans nous préoccuper des délais introduits par le MCCP et leur influence sur le niveau de qualité de service du crypto-système. Dans le cas de l'ordonnancement FIFO décrit précédemment, le délai de traitement par le MCCP d'un paquet entrant peut être déterminé de la façon suivante : si l'on considère que plusieurs paquets provenant de canaux

différents peuvent arriver en même temps en entrée du processeur hôte alors le délai maximum de traitement d'un paquet par le MCCP peut être borné par l'inégalité suivante :

$$d_k \leq \max_{i \in [0,k-1]} (e_i) \times \left\lfloor \frac{k}{m} \right\rfloor + e_k$$

Où d_k est le délai de traitement du $k^{ème}$ paquet en attente, e_i le temps de traitement du $i^{ème}$ paquet et m le nombre de cœurs cryptographiques embarqués dans le MCCP. Bien que cette inégalité borne de façon large le délai de traitement du paquet k, celle-ci illustre bien le fait que le délai entre l'arrivée d'un paquet et la fin de son traitement *augmente proportionnellement* avec le nombre de canaux actifs. En effet, si l'équation (4) est vérifiée alors le nombre maximum de paquets en attente de traitement est égal au nombre de canaux ouverts à un instant donné. Il s'ensuit que si les types des paquets traités par le MCCP sont hétérogènes (ex. tailles de paquet différentes, latences maximums admissibles différentes, etc.) alors un algorithme d'ordonnancement FIFO peut poser des problèmes de qualité de service. À titre d'exemple, le chiffrement d'un paquet de 2 Ko sur un cœur avec l'algorithme AES-CCM nécessite 76 µs à 190 MHz. Dans le cas où 32 canaux seraient ouverts sur un MCCP embarquant deux cœurs, les délais de traitement d'un paquet pourraient dépasser la milliseconde. Si un temps de traitement d'une milliseconde semble faible par rapport au délai maximum de transmission admissible qui est de l'ordre de 150 ms pour la vidéoconférence, il faut garder à l'esprit que le crypto-système n'est qu'une sous-partie d'une radio logicielle complète. En outre, l'étude précédente ne tient pas compte des délais introduits par le fonctionnement du processeur hôte.

Étant donné que cette étude ne cible pas une application particulière de la radio logicielle, il n'est pas possible d'estimer un délai maximum acceptable pour le traitement d'un paquet de données. Toutefois, dans le cas où un MCCP ne répondrait pas aux besoins en termes de latence d'une radio logicielle, les concepteurs du système auraient le choix entre augmenter le nombre de cœurs embarqués dans le MCCP (ou leur performance) ou mettre en œuvre un mécanisme d'ordonnancement des paquets plus évolué que la simple FIFO (cf. section 3.6.2).

4.6 Application de la reconfiguration partielle

4.6.1 Plateforme de test de la reconfiguration dynamique partielle

Avant d'aborder le cas de la reconfiguration du MCCP, cette partie présente quelques travaux portant sur la reconfiguration partielle des FPGA. Ces travaux ont permis d'aborder les problématiques liées aux outils et aux techniques de reconfiguration partielle. En effet, bien que la reconfiguration partielle soit disponible sur les FPGA Xilinx depuis plusieurs années, la commercialisation de cette technologie n'a commencé que depuis peu ce qui fait qu'à cette heure les outils disponibles ne sont pas complètement aboutis. Il s'ensuit que l'implantation d'un circuit dynamiquement reconfigurable relève parfois de la gageure.

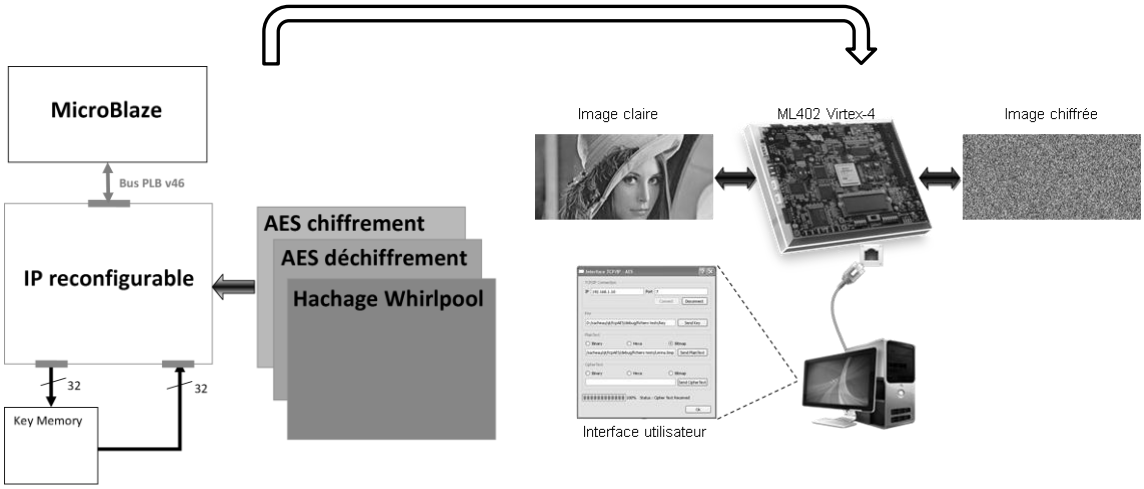


Figure 48 : Plateforme d'évaluation de la reconfiguration sur Virtex 4

Ces travaux ont porté sur l'implantation de circuits cryptographiques dynamiquement reconfigurables sur une carte ML402 équipée d'un FPGA Virtex 4 SX35. La plateforme de test utilisée était composée d'un processeur Xilinx *MicroBlaze* chargé du contrôle de l'accélérateur cryptographique implanté et du module ICAP de la plateforme. Cette plateforme de test était elle-même contrôlée par le biais un PC hôte connecté à la carte d'évaluation par l'intermédiaire d'une connexion Ethernet (cf. Figure 48).

Tableau 20 : Occupation en ressource de la plateforme de test sur V4 SX35

Circuit		LUT	FF	SLICE	DSP48	RAM16
Partie statique		15443	9060	10764	3	97
Partie dynamique	**Chiffrement**	481	293	351	0	4
	Déchiffrement	558	298	395	0	4
	Whirlpool	2008	309	1153	0	4

L'application développée consiste à charger dynamiquement sur le FPGA des modules de chiffrement AES, de déchiffrement AES et de hachage Whirlpool. Le Tableau 6 donne l'occupation en ressource de cette application. On peut remarquer que les modules de chiffrement et déchiffrement occupent approximativement le même nombre de slices alors que l'implantation du module de hachage nécessite environ quatre fois plus de slices. Pour que la zone dynamiquement reconfigurable soit compatible avec l'ensemble des modules, elle doit être au moins de la taille du plus grand des modules.

Si, dans un contexte d'évolution du système sur le long terme, on ne connait pas à l'avance l'ensemble des modules qui seront dynamiquement configurés, il est nécessaire de sur-dimensionner cette zone. Dans le cas de l'application testée, le plus grand des modules nécessite un nombre de slice qui représente moins de 8% du nombre de slices disponible dans le FPGA ciblé (FPGA V4 SX35). Or, ce type de FPGA est loin d'être le circuit ayant la plus forte densité d'intégration du fait sa technologie SRAM 90 nm. La taille de la zone dynamiquement reconfigurable peut donc paraitre comme une contrainte souple.

Toutefois, il ne faut pas oublier que le temps de reconfiguration de cette zone dépend de la taille du bitstream et donc de la taille de cette zone. De plus, la taille de la mémoire de stockage des bitstreams partiels est elle aussi un élément à prendre en compte.

Afin d'illustrer ces observations, le Tableau 21 présente les performances temporelles de notre plateforme de reconfiguration. Les deuxième et troisième colonnes de cette table contiennent les délais nécessaires à la reconfiguration des différents bitstreams partiels avec un circuit Virtex 4. Si l'on prend en compte la latence maximum de 150 ms pour la vidéoconférence, on constate qu'il n'est pas possible de reconfigurer à la volée la plateforme pour ce type d'application. Une solution à ce problème est de faire fonctionner l'ICAP au maximum de sa bande passante en stockant les bitstreams partiels dans les mémoires embarquées dans le FPGA (BRAM). On obtient alors les temps de reconfiguration qui sont donnés dans la quatrième colonne du Tableau 7. Malheureusement, le stockage des bitstreams partiels dans les BRAM du FPGA n'est pas envisageable car leur nombre est très limité. Par exemple, le stockage dans le FPGA des trois bitstreams partiels de notre application utilise 80 % des BRAM disponibles sur le Virtex 4.

Tableau 21 : Taille et temps de reconfiguration des différents bitstreams partiels sur V4 SX35

Bitstream partiel	Taille du fichier de bitstream partiel (ko)	Temps de reconfiguration avec lecture depuis CF (ms)	Temps de reconfiguration avec lecture depuis RAM (ms)	Temps de reconfiguration minimum (µs)
chiffrement	89	380,6	63,5	222
déchiffrement	96	413,4	68,9	239
blank	66	281,6	46,75	164
hachage	97	416,6	69,1	242

L'utilisation d'un algorithme de compression permet éventuellement de diminuer fortement la taille des bitstreams partiels. Cependant, les outils proposés par les fabricants de FPGA ont des performances limitées qui obligent l'utilisateur à implanter son propre module de compression [GuCh08]. Quoi qu'il en soit, la taille des bitstreams compressés reste suffisamment importante (plusieurs Ko) pour dissuader de les stocker en BRAM. Cela pousse à envisager des solutions alternatives passant par l'utilisation de systèmes spécifiques basés sur des mémoires RAM externes. Toutefois, ce type de solution pose problème dans le cadre la radio logicielle sécurisée dans la mesure où elle nécessite de protéger la mémoire externe, cette protection ayant un coût en ressources et en performances (débit/latence). En définitive, vu les éléments à notre disposition, il semble que la seule solution viable et sécurisée à ce problème soit d'embarquer davantage de mémoire à l'intérieur des FPGA. De plus amples informations sur les différentes techniques de reconfiguration partielle et leurs performances sont disponibles dans [LKLJ09c].

4.6.2 **Application de la reconfiguration dynamique partielle au MCCP**

Les performances générales de la reconfiguration partielle dans le cadre de la cryptographie ayant été abordées, la suite de cette partie aborde la question de l'utilisation de la reconfiguration partielle pour le MCCP précédemment présenté. Il s'agit ici de tirer profit des possibilités offertes par la reconfiguration partielle tout en limitant les modifications qui devront être apportées à l'architecture déjà implantée. Le Tableau 19 montre que le débit maximum avec le mode de chiffrement CCM est obtenu lorsque deux cœurs sont utilisés en parallèle pour traiter un même paquet de données. L'inconvénient de cette configuration réside dans le fait qu'elle nécessite l'utilisation partielle de deux cœurs de calcul, le module GHASH n'étant pas utilisé. Afin d'améliorer le niveau d'utilisation des ressources disponibles, il est possible de rendre dynamiquement reconfigurable la zone embarquant le module de hachage GHASH. La Figure 49 illustre les modifications apportées à l'architecture : le module GHASH/AES est maintenant connecté au cache de clé du cœur cryptographique ; à cet effet un bloc RAM supplémentaire est utilisé. Aucune autre modification autre que l'ajout du module ICAP aux périphériques du processeur hôte n'est nécessaire dans un premier temps. Par la suite, il sera toutefois nécessaire de connecter directement le module ICAP au *contrôleur principal* du MCCP. Par ailleurs, le module AES déjà implanté peut être réutilisé comme module partiellement reconfigurable, seules quelques modifications visant à uniformiser les ports d'E/S présents sur les interfaces de communication des modules AES et GHASH sont nécessaires.

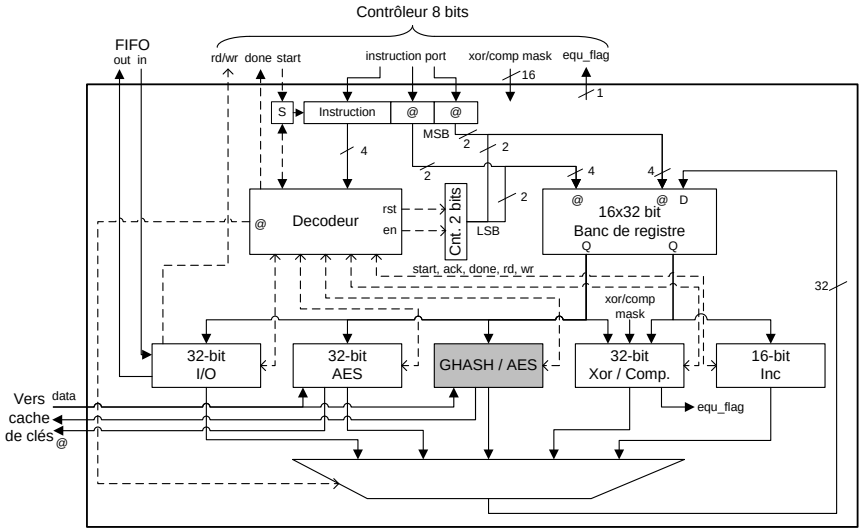


Figure 49 : Unité cryptographique dynamiquement reconfigurable

Une fois le code VHDL modifié, il est possible d'utiliser deux modules de chiffrement AES fonctionnant en parallèle dans un unique cœur ce qui donne les performances décrites dans le Tableau 14 page 102.

Sur Virtex 6 LX240T-1, entre quatre et six *frames* de FPGA sont nécessaires à la mise en place d'une zone partiellement reconfigurable supportant les modules AES et GHASH. La configuration à six *frames* embarque les blocs RAM destinés au calcul de l'opération SubBytes d'AES, la mémoire de configuration contenue dans ces six *frames* est de 97 Ko. Avec la configuration à quatre *frames,* les blocs RAM sont implantés à l'extérieur de la zone reconfigurable, cette solution utilise une mémoire de configuration de 46 Ko. Si l'on adopte la deuxième solution, le temps de reconfiguration théorique minimum est de 116 µs à 100 MHz. Une fois encore, il n'est pas possible de donner une réponse définitive à la question qui est de savoir si ce délai de reconfiguration est excessif ou non. Toutefois, si l'on compare ces 116 µs avec les 150 ms de latence maximum pour la voix et la vidéo « temps réel », on peut estimer que le rapport de 1 pour 1000 permet d'envisager son utilisation pour ce genre d'application tant que le nombre de reconfiguration par seconde reste faible.

4.7 **Comparaison avec l'état de l'art**

Afin de comparer les capacités du MCCP avec celles des crypto-systèmes publiés dans la littérature, nous reprendrons ici la même approche que celle adoptée pour la synthèse du chapitre Chapitre 1 (cf. partie 2.4.6 page 71). Nous comparerons donc dans cette partie le MCCP avec les architectures *Celator*, *COBRA*, *CrCU*, *Cryptomaniac* et *AESTHETIC* au travers des cinq critères d'évaluation que sont *l'interopérabilité*, *l'évolutivité*, les *performances* et le *parallélisme*.

Interopérabilité : L'interopérabilité du MCCP est basée sur le fait que celui-ci possède une architecture à la fois *programmable* et *dynamiquement reconfigurable*. Le caractère reconfigurable du MCCP s'apparente à celui de *CrCU* dans la mesure où ces deux crypto-systèmes sont destinés à être implantés sur FPGA. Néanmoins, le MCCP possède l'avantage par rapport à CrCU de pouvoir être reprogrammé afin d'exécuter un mode de chiffrement différent. Chose qu'il n'est pas possible de faire avec le second qui est basé sur une architecture dédiée. Par contre, l'avantage du MCCP en termes d'interopérabilité face à *Celator*, *COBRA* ou encore *Cryptomaniac* est plus limité. En effet, ces crypto-systèmes possèdent des chemins de données reconfigurables et/ou programmables. Dans ces conditions, nul besoin d'avoir recours à la reconfiguration dynamique partielle afin d'être compatible avec un grand nombre de protocoles de chiffrement : le passage d'un algorithme de chiffrement à un autre en est donc facilité et accéléré. On notera toutefois que l'avantage de ces crypto-systèmes est contrebalancé par des performances moindres. Quant à AESTHETIC, le fait que ce crypto-système soit spécifique à AES lui confère un niveau d'interopérabilité très faible.

Evolutivité : Tout d'abord, *AESTHETIC*, *COBRA*, *Celator* et *Cryptomaniac* ont été conçus pour être implantés sur ASIC ce qui fait que leur capacité à évoluer est nulle, contrairement au MCCP qui est conçu pour être implanté sur FPGA. Pourtant, on pourra remarquer que les opérateurs implantés dans *COBRA*, *Celator* et *Cryptomaniac* sont suffisamment génériques pour rendre probable la compatibilité de ces architectures avec de futurs algorithmes de chiffrement. De plus, un portage sur FPGA à l'image de *CrCU* peut tout à fait être

envisagé. Dans les faits, le principal atout de MCCP par rapport à ces architectures réside dans sa capacité à être *reconfiguré dynamiquement* (tout comme *CrCU* dans sa version dynamiquement reconfigurable). Dans ces conditions, il n'est donc plus forcément nécessaire d'interrompre le fonctionnement du système pour le faire évoluer. Le MCCP a donc la capacité d'être modifié à tout moment et ce même en cours de fonctionnement. Cette capacité est toutefois à relativiser dans la mesure où la structure du FPGA en elle-même est figée.

Performances : Comparer de façon équitable les performances des différents crypto-systèmes dont l'architecture a déjà été publiée avec celles du MCCP n'est pas une tâche aisée. En effet, si les performances d'un composant dépendent bien de son architecture, elles dépendent aussi de la cible technologique utilisée pour son implantation. De plus, le rapport du débit sur la surface occupée ne devrait pas non plus être négligé. Il s'ensuit que ce passage n'a pas pour objectif d'être une comparaison objective et équitable des performances du MCCP mais plutôt un aperçu du positionnement des performances du MCCP par rapport aux architectures de crypto-systèmes existantes. Avec un débit de 2,6 Mbps/MHz par cœur de calcul (soit 10,4 Mbps/MHz pour quatre cœurs) dans le mode de chiffrement ECB, le MCCP se trouve dans la fourchette haute des crypto-systèmes (cf. Tableau 4 page 61). Néanmoins, les crypto-systèmes dédiés dont la structure est déroulée et massivement pipeline sont bien plus performants que le MCCP en ce qui concerne le chiffrement de flux de données dans le cadre d'une application monostandard.

Parallélisme : Par parallélisme, nous entendons ici la capacité d'un crypto-système à gérer de façon concurrente plusieurs canaux de communication. Dans ces conditions, l'amélioration du niveau de parallélisme d'un crypto-système passe par la diminution des délais de changement de clés et d'algorithmes et l'augmentation du nombre de messages traités de façon concurrente. Comme *CrCU*, *COBRA* ou *AESTHETIC*, le MCCP est capable de générer matériellement les sous-clés de chiffrement ce qui d'une part décharge le processeur hôte de ces calculs et d'autre part diminue les délais de génération de ces clés. De plus, le MCCP permet de générer les sous-clés *à la volée* ce qui diminue encore les délais d'attente nécessaires induits par le changement de la clé chiffrement. En ce qui concerne le nombre de paquets qui peuvent être traités de façon simultanée, le MCCP tout comme *AESTHETIC*, *CrCU* ou encore *Cryptomaniac* tire profit de sa structure multicœur. Mais contrairement à *AESTHETIC* ou *CrCU*, la distribution des tâches aux cœurs de calcul est gérée par le MCCP lui-même. Cela permet d'une part d'améliorer l'efficacité de ce processus mais aussi de décharger de cette tâche le processeur hôte. Le MCCP propose donc les mêmes avantages que *Cryptomaniac*.

Sécurité : En ce qui concerne la sécurité, le MCCP possède plusieurs avantages par rapport aux crypto-systèmes cités dans la mesure où il a été conçu dès le départ dans l'objectif d'être sécurisé. D'abord, contrairement aux autres crypto-systèmes présentés, le MCCP embarque donc une mémoire de stockage sécurisé des clés de chiffrement dont il est le seul à avoir un accès en lecture. De plus, cette mémoire est directement connectée aux primitives cryptographiques qui sont conçues de manière à éviter toute fuite des clés de

chiffrement en dehors d'elle. Ensuite, le fonctionnement du MCCP étant autonome, son isolation par rapport au reste du système s'en trouve améliorée, ce qui tend à rendre plus difficile la mise en œuvre d'attaque logicielle contre celui-ci. Pour finir, le fait que le MCCP soit destiné à être implanté sur FPGA permet au concepteur de le mettre à jour afin d'y appliquer d'éventuelles mises à jour de sécurité (possibilité qu'il partage avec *CrCU*).

4.8 **Conclusion**

Dans ce chapitre, l'architecture et l'implantation d'un MCCP performant ont été présentées. Les problématiques de débit, de la latence, de l'occupation des ressources et de la reconfiguration partielle ont été abordées et quelques conclusions ont pu être tirées des résultats obtenus. Plus précisément, nous avons démontré qu'un système aussi complexe qu'un MCCP n'occupe finalement qu'une petite partie des FPGA les plus récents. Nous avons aussi montré que les performances en termes de débit du MCCP implanté sont largement suffisantes pour la voix ou la vidéo en « temps réel ». En ce qui concerne l'ordonnancement des tâches ou encore la reconfiguration partielle, il n'existe pas de réponse toute faite adaptée à toutes les conditions. En ce sens, les résultats présentés dans ce chapitre ont pour intérêt de donner au lecteur les clés lui permettant d'évaluer les capacités et les limites de notre approche. L'intérêt de cette évaluation réside dans le fait qu'elle rende possible d'estimer, a priori, les avantages et les inconvénients induits par l'utilisation du MCCP. Si l'on reprend l'exemple de la reconfiguration partielle, il est évident que celle-ci permet d'accroître la flexibilité du système au détriment de la latence et de la complexité. En revanche, l'apparition d'IP matérielle de FPGA ou eFPGA (*embedded* FPGA) laisse envisager l'implantation sur ASIC du MCCP tout en conservant l'aspect reconfigurable. De cette manière, la matrice FPGA pourrait être optimisée en prenant en compte les besoins du MCCP de manière à réduire la complexité de la matrice de configuration ainsi que son temps de reconfiguration.

Conclusion et perspectives

Conclusion et critique des résultats

Depuis une dizaine d'année, le concept de la radio logicielle est mis en œuvre au sein d'architectures hétérogènes sensées répondre à l'explosion du nombre de standards de communication. Les nouvelles architectures ainsi développées sont particulièrement utiles aux militaires pour lesquelles interopérabilité et flexibilité sont qualités indispensables. Cependant, les applications militaires nécessitent la mise en place de mécanismes de sécurité dont les spécifications sont absentes de standard tel que la *Software Communication Architecture*. Afin de remédier à cela, le *supplément sécurité à la SCA* s'attache à formaliser et à standardiser les mécanismes indispensables à la *sécurisation* d'une radio logicielle. Dans ce document, l'architecture d'un composant sécurisé conforme au supplément sécurité à la SCA et l'architecture d'un crypto-système qui pourrait y être intégré sont présentés. Plus précisément, nous nous demandions dans l'introduction qu'elle pourrait être la forme d'un *sous-système cryptographique* implanté sur FPGA. C'est-à-dire : comment exploiter efficacement les structures massivement parallèles des FPGA et tirer profit de leur capacité à être reconfigurés, tout en garantissant la sécurité du *sous-système cryptographique* ?

La réponse que nous apportons à cette question est détaillée tout au long des quatre chapitres de ce document. La première partie du Chapitre 1 concerne l'étude bibliographique de la radio logicielle et du standard SCA, permettant ainsi de spécifier le contexte de ce travail. S'appuyant sur cette étude bibliographique, une architecture de *sous-système cryptographique* compatible avec le supplément sécurité à la SCA, et destinée à être implantée sur FPGA, est présentée dans la seconde partie du chapitre. En outre, il se trouve que le processus de recherche ayant mené à la mise au point de cette architecture a montré l'intérêt d'étudier plus en profondeur la structure et le fonctionnement du ou des composants cryptographiques intégrés dans le *bloc de communication* du *sous-système cryptographique*.

Le Chapitre 2 présente donc logiquement un état de l'art des travaux publiés dans le domaine des crypto-systèmes. Cet état de l'art débute par une description des caractéristiques que doit posséder un crypto-système dédié à la radio logicielle. Dans la suite du chapitre, cette description sert de base pour évaluer l'adéquation des différentes classes de crypto-systèmes avec les besoins de la radio logicielle. Une étude détaillée de quelques architectures de crypto-systèmes permet d'illustrer concrètement ces propos. Cet état de l'art conclut à l'inadéquation partielle des crypto-systèmes existants avec les besoins de notre CSS et plus généralement avec ceux de la radio logicielle.

Le Chapitre 3 fait suite aux conclusions du Chapitre 2 en présentant une architecture théorique de cryptoprocesseur multicœur reconfigurable (MCCP). Sont donc tour à tour

abordées les questions relatives à l'interfaçage du cryptoprocesseur, son fonctionnement, la structure de ses cœurs ou encore l'ordonnancement des tâches qu'il exécute. Il s'ensuit que ce chapitre présente un large éventail de mécanismes permettant au MCCP qui les utilise de répondre aux besoins de la radio logicielle.

Afin d'illustrer concrètement les propos du précédent chapitre, le Chapitre 4 détaille la mise au point d'un prototype de MCCP implanté sur Virtex 4 et Virtex 6 (cf. Chapitre 4). En proposant des débits d'environ 10 Mbps/MHz notre cryptoprocesseur se classe parmi les architectures *multistandard* les plus performantes dans la mesure où seules les architectures dédiées proposent des débits supérieurs. Ce niveau de performance et d'interopérabilité est rendu possible par l'utilisation des fonctions de *reconfiguration dynamique partielle* des FPGA modernes. En effet, ces dernières permettent la mise au point de composants matériels aussi flexible que des composants logiciels sans pour autant souffrir des mêmes limitations en termes de performances.

On pourra toutefois remarquer que les travaux développés dans ce document consistent plus en une méthodologie de conception de composants sécurisés pour la radio logicielle qu'en la mise en œuvre d'une solution complète à ce problème. En effet, la diversité des cas d'utilisation des radios logicielles ajouté au fait que les algorithmes cryptographiques militaires sont classifiés ont fait qu'il ne nous a pas été possible d'illustrer nos travaux par un exemple concret d'implantation du MCCP dans une radio logicielle. En outre, étant donné les limites de temps inhérentes au déroulement des travaux de thèse, nous n'avons implanté que partiellement sur FPGA les différentes propositions que l'on retrouve dans ce document. Quoiqu'il en soit, ces travaux ont tout de même conduit au développement d'un certain nombre de ressources et notamment d'IP matérielles librement disponibles à la communauté scientifique. Pour finir, trois sujets méritent d'être approfondis : l'ordonnancement des tâches cryptographiques sur le MCCP, la structure des circuits d'interconnexion entre les cœurs cryptographique et enfin la technologie eFPGA dans le cadre des composants cryptographiques sécurisés. Ces sujets sont donc succinctement décrits dans la partie suivante.

Perspectives et pistes de travail

Au cours de ce document de nombreux aspects de la conception d'un MCryptoPSoC (*Multi CryptoProcessor System on Chip*) ont été abordés. Cependant, au moins deux d'entre eux n'ont été traités que partiellement : *l'ordonnancement des tâches* et *l'interconnexion des cœurs sur FPGA*. L'étude approfondie de ces deux sujets aurait un réel intérêt scientifique dans la mesure où cela permettrait de compléter, voire même d'élargir, le cadre d'application de nos travaux.

Ces réflexions sur les réseaux d'interconnexions ont amenés à envisager l'utilisation de composant FPGA embarqués appelés eFPGA pour *embedded FPGA*. Cette technologie possède de sérieux atouts en ce qui concerne l'implantation de composants cryptographiques à haut niveau de sécurité. En effet, l'utilisation de composant eFPGA dans des systèmes sur puce permet d'intégrer autant de composants spécifiques à un SoC

que nécessaires. En outre, dans ces conditions, l'implantation de contremesures de bas niveau (ex. lissage de la consommation d'énergie) n'est plus un problème. Pour finir, il est possible d'améliorer la sécurité et la fiabilité d'un SoC en circonscrivant la possibilité de reconfigurer un circuit aux seules zones qui en ont impérativement besoins (ex. primitives cryptographiques).

En conclusion, de nombreuses et intéressantes questions restent en suspens quant à la manière d'utiliser efficacement les architectures reconfigurables pour l'implantation de circuits sécurisés.

Annexe A

Cahier des charges du CSS

A.1 Fonctions du CSS

A.1.1 Fonctions principales

- Fonctions cryptographiques :
  - Chiffrement/Déchiffrement des données émises/reçues par la radio.
  - Fonctions TRANSEC (**non prises en compte ici**).
  - Génération et vérification des signatures.
  - Contrôle d'intégrité.
- Bypass de données :
  - Données en clair émises/reçues par la radio.
  - Données de contrôle/statut entre les zones rouge et noire.

A.1.2 Fonctions nécessaires

Afin d'implémenter les fonctions principales du CSS, les fonctions suivantes doivent être disponibles :

- Initialisation, fonctionnement et arrêt :
  - Boot
  - Instanciation
  - Fonctionnement
  - Arrêt prévu
  - Arrêt imprévu
- Isolation R/N
- Fonctions cryptographiques :
  - Chiffrement/Déchiffrement
  - Identification/Authentification
  - Intégrité
- Fonctions de gestion sécurisée des ressources :
  - Gestion des clés
  - Gestion des algorithmes (logiciel et bitstream)
  - Gestion des polices de sécurité
- Fonction de bypass :
  - Communications en clair
  - Données de contrôle/statut

- Interfaces cryptographiques

A.2 Cahier des charges du CSS

A.2.1 Contraintes Générales

1. Le CSS doit être multicanal
2. Le CSS doit être protégé contre les attaques matérielles.
3. Le nombre maximal de canaux cryptographiques disponibles doit être constant.
4. Le bloc de communication doit fournir un débit de l'ordre du Mbit/s.
5. La taille du chemin de données doit permettre une implantation aisée des IP cryptographiques.

A.2.2 Contraintes à l'initialisation, au fonctionnement et à l'arrêt

A.2.2.1 *Contraintes au boot*

1. Le CSS doit utiliser un système de boot contenu à l'intérieur de l'enceinte cryptographique.
2. Le CSS doit vérifier la validité des données stockées dans les mémoires non-volatiles.
3. Le CSS doit effectuer des tests internes pour s'assurer de son bon fonctionnement.
4. Le CSS devrait être initialisé avant que les formes d'ondes ne soient créées.

A.2.2.2 *Contraintes à l'instanciation (des canaux)*

1. Le CSS doit vérifier les droits de la forme d'onde avant d'instancier un algorithme cryptographique.
2. Le CSS doit déchiffrer et instancier les algorithmes cryptographiques lors de l'instanciation de la forme d'onde.
3. Le CSS doit tester le bon fonctionnement des algorithmes avant de permettre le fonctionnement de la forme d'onde.
4. La radio (*DomainManager*) doit fournir les données de connexion aux formes d'ondes.
5. Le CSS doit permettre la sélection d'une clé particulière.
6. Le CSS doit vérifier que les clés, algorithmes et formes d'ondes ont le même niveau de classification lorsqu'ils sont utilisés ensemble.
7. Le CSS doit fournir un système de génération de nombre aléatoire pour l'initialisation des algorithmes cryptographiques.

A.2.2.3 *Contraintes de fonctionnement*

1. Le CSS doit effectuer de façon périodique des tests de bon fonctionnement. Il s'agit par exemple d'assurer l'intégrité de la configuration du FPGA.
2. Le CSS doit fournir son statut à un utilisateur autorisé.

A.2.2.4 *Arrêt prévu*

1. Lors de la fermeture d'un canal, le CSS doit effacer les données cryptographiques utilisées.
2. Lors de la fermeture d'un canal, le CSS doit effacer les clés cryptographiques utilisées.
3. Lors de la fermeture d'un canal, le CSS doit effacer les algorithmes cryptographiques utilisés.
4. Le CSS doit notifier la fermeture d'un canal sur sa sortie de statut.

A.2.2.5 *Arrêt imprévu*

1. A l'occurrence d'une erreur interne, le CSS doit entamer une procédure de sécurité pour terminer/suspendre les opérations en cours.
2. Le CSS doit contrôler l'apparition d'alarmes extérieures et déclencher une procédure spécifique le cas échéant.
3. Le CSS devrait notifier son statut lors d'une erreur.
4. Le CSS ne doit pas déclencher de procédure automatique de rétablissement lorsqu'une erreur est détectée.
5. Le CSS doit accepter les demandes de reset effectuées par un utilisateur autorisé.

A.2.3 **Fonctions de confidentialité**

A.2.3.1 *Contraintes de chiffrement*

1. Le CSS doit permettre le chiffrement de données transitant depuis la zone rouge vers la zone noire.
2. Si besoin, le CSS doit pouvoir chiffrer des logiciels ou des données pour le compte de la radio en vue de leur stockage.
3. Le CSS doit pouvoir chiffrer des clés pour le compte de la radio en vue de leur stockage.
4. Le CSS doit pouvoir utiliser plusieurs algorithmes cryptographiques différents.
5. Le CSS doit pouvoir chiffrer des données en provenance de la zone rouge et les retourner à la zone rouge.
6. Le CSS doit pouvoir chiffrer des données en provenance de la zone noire et les retourner à la zone noire.

A.2.3.2 *Contraintes de déchiffrement*

1. Le CSS doit permettre le déchiffrement de données transitant depuis la zone noire vers la zone rouge.
2. Si besoin, le CSS doit pouvoir déchiffrer des logiciels ou des données depuis une mémoire de stockage interne à la radio.
3. Le CSS doit pouvoir déchiffrer des clés depuis une mémoire de stockage interne à la radio.
4. Le CSS doit pouvoir utiliser plusieurs algorithmes cryptographiques différents.
5. Le CSS doit pouvoir déchiffrer des données en provenance de la zone rouge et les retourner à la zone rouge.

6. Le CSS doit pouvoir déchiffrer des données en provenance de la zone noire et les retourner à la zone noire.

A.2.4 **Fonctions d'identification et d'authentification**

Elles sont utilisées pour authentifier et identifier la provenance des messages échangés par la radio. Mais aussi pour contrôler les accès au CSS.

1. Le CSS doit effectuer une authentification cryptographique des messages R et N.
2. Le CSS doit retourner le résultat de l'authentification au demandeur.
3. Le CSS doit fournir des mécanismes de signature des messages.
4. Le CSS doit permettre le stockage de certificats.

A.2.5 **Fonctions d'intégrité**

L'intégrité des données peut être vérifiée de manière cryptographique ou non suivant le niveau de confidentialité des données traitées.

1. Le CSS doit vérifier l'intégrité des messages R et N de façon cryptographique.
2. Le CSS doit retourner le résultat du contrôle d'intégrité au demandeur.
3. Le CSS doit fournir un service de contrôle d'intégrité utilisant SHA.
4. Le CSS doit fournir un service de contrôle d'intégrité standard (parité, CRC, …)

A.2.6 **Fonctions de gestion sécurisées**

A.2.6.1 ***Contraintes de gestion des clés***

Est appelé clé toute clé cryptographique, certificat, accréditation ou vecteur d'initialisation. La sécurité du réseau entier dépend de la sécurité des clés.

La gestion des clés comprend :

- Le chargement des clés et leur identification
- Le stockage des clés
- L'instanciation et l'utilisation des clés
- L'effacement des clés
- La génération de clés

A.2.6.2 ***Réception et identification des clés***

1. Le CSS doit pouvoir spécifier des attributs et un ID aux clés.
2. Le CSS doit permettre à un opérateur authentifier d'ajouter des clés au système.
3. Les attributs et ID des clés doivent être accessibles à un opérateur authentifié.
4. Le CSS doit permettre le chargement de clé noire en conservant les autres canaux opérationnels.

A.2.6.3 ***Stockage des clés***

1. Les clés doivent être stockées de manière chiffrée.

2. Les clés rouges ne doivent pas sortir du CSS.
3. Les clés doivent être stockées avec leurs données d'identifications.

A.2.6.4 *Instanciation et utilisation des clés*

1. Les demandes de clé doivent se faire par l'intermédiaire des identifiant de clé.
2. Les clés ne doivent pas pouvoir être instanciées lorsque :
    a. La clé n'a pas le bon niveau de classification
    b. Le type de clé n'est pas le bon
    c. Le type de la clé ne correspond pas au type de clé supporté par l'algorithme
3. L'intégrité des clés doit être testée à l'instanciation
4. Le CSS doit permettre aux clés noires de sortir du CSS.
5. Le CSS doit pouvoir détecter une altération des clés.

A.2.6.5 *Effacement des clés*

1. Le CSS doit permettre l'effacement matériel et logiciel des clés rouges.
2. Le CSS doit permettre d'effacer de façon sélective les clés rouges ou noires.
3. L'effacement des clés doit être authentifié.
4. La mise à jour d'une clé doit effacer sa valeur antérieure.

A.2.6.6 *Gestion des clés*

1. Le CSS doit maintenir un fichier spécifiant le propriétaire de la clé, la forme d'onde, l'algorithme et le réseau auxquelles elle est associée.
2. Un journal des clés chargé, remis à zéro et effacé doit être maintenu.
3. Le CSS doit informer la radio lorsqu'une clé expire.

A.2.7 **Contraintes sur la gestion des algorithmes**

Sont considérés comme algorithmes les algorithmes cryptographiques implémentés sous forme de logiciels ou de bitstreams.

A.2.7.1 *Chargement et identification des algorithmes*

1. Les algorithmes classifiés doivent être chiffrés.
2. Le CSS doit vérifier l'intégrité et l'authenticité des algorithmes chargés.
3. Le résultat du contrôle d'intégrité et d'authenticité des algorithmes doit être enregistré dans un journal.
4. Le CSS ne doit pas accepter de charger un algorithme plus ancien que la plus récente version disponible.
5. Un opérateur habilité doit pouvoir charger de nouvelles versions des algorithmes.

A.2.7.2 *Stockage des algorithmes*

1. Les algorithmes doivent être stockés sous forme chiffrée.
2. Chaque algorithme doit être identifié par un type et une version.

A.2.7.3 *Instanciation et utilisation des algorithmes*

1. Les demandes d'instanciation d'algorithmes doivent être acceptées en fonction des besoins des formes d'onde.
2. Les algorithmes instanciés doivent être testés avant d'être utilisés.
3. Les algorithmes doivent être effacés à la fermeture du canal.
4. Dans une radio multicanal, un nouvel algorithme doit pouvoir être instancié alors que d'autres canaux sont déjà opérationnels.
5. L'algorithme doit avoir un accès restreint aux ressources. En particulier il ne doit pas être possible de charger une IP n'importe où sur le FPGA.

A.2.7.4 *Effacement des algorithmes*

1. Les anciennes versions des algorithmes doivent être effacées lorsqu'une version plus récente est chargée.

A.2.8 **Gestion des polices de sécurité**

Les règles de sécurité internes du CSS spécifient :

- Les conditions de déclenchement des alarmes et le cas échéant, les procédures à suivre.
- Les moyens de communication autorisés avec l'extérieur du CSS.
- Les conditions de communication entre les composants internes au CSS.
- Les conditions d'interconnexion des algorithmes avec les formes d'onde.
- Le niveau de classification du matériel.

1. Les règles de bypass doivent être protégées soit en étant stockées dans le CSS soit en contrôlant leur authenticité et leur intégrité.
2. Un utilisateur habilité doit pouvoir charger des règles de sécurité dans le CSS.
3. Le CSS doit pouvoir fournir un statut concernant les règles de sécurité à un utilisateur habilité.
4. Les tables de règles de sécurité ne doivent pas être accessibles sans authentification.
5. La radio doit pouvoir charger les règles de sécurités concernant l'affectation des algorithmes aux formes d'onde.

A.2.9 **Sécurité des logiciels critiques**

1. Le CSS doit déchiffrer les logiciels critiques stockés en interne à la radio.
2. Le CSS doit chiffrer les logiciels critiques stockés dans un module interne à la radio.
3. L'intégrité des logiciels critiques doit être vérifiée.
4. L'intégrité des logiciels critiques doit être garantie durant leur fonctionnement.

A.2.10 **Contraintes sur le module de bypass**

Le module de bypass traite deux types d'informations. Les informations reçues/émises en clair par la radio et les informations de contrôle/statut échangées en clair entre les zones rouge et noire.

A.2.10.1 ***Bypass des informations de contrôle/statut***

1. Le module de bypass doit être non contournable.
2. Les paramètres de bypass doivent être associés à l'application les requérant.
3. Le système de bypass des informations de contrôle/statut doit être distinct du système de bypass utilisé pour les messages radio.
4. Le mécanisme de bypass doit notifier toutes infractions aux règles de sécurité.
5. Le module de bypass doit bloquer les messages en infraction.
6. Si le nombre d'infractions aux règles de sécurité dépasse un certain seuil, le canal correspondant doit être fermé.
7. Le module de bypass doit vérifier les connexions entre objets, la validité, la taille et la fréquence des messages.
8. La police est téléchargée au format XML.
9. La police doit être stockée à l'intérieur du CSS ou protégée en accès et son intégrité garantie.

A.2.10.2 ***Bypass des données radios***

1. Le module de bypass doit être non contournable.
2. L'*ApplicationFactory* doit fournir au CSS les polices de sécurité du module de bypass.
3. Le CSS doit fermer un canal lorsqu'une infraction aux polices de sécurité est détectée.

A.2.11 **Contraintes sur le contrôle du CSS par la radio**

1. Le CSS doit accepter des messages de contrôle depuis des sources habilitées.
2. Le CSS doit accepter les fichiers définissant les interconnexions entre le CSS et les formes d'onde.
3. CSS doit stocker de façon interne les fichiers définissant ces interconnexions.
4. Le CSS doit pouvoir fournir des informations de statut et d'alarme.
5. Le CSS doit pouvoir prendre en compte les alarmes générées par la radio.
6. Le CSS doit maintenir des polices de sécurité déterminant la réponse à une alarme interne ou externe au CSS.

A.2.12 **Contraintes sur les processus**

A.2.12.1 ***Protection des logiciels***

1. Le comportement d'un logiciel ne doit pas pouvoir être modifié en cours de fonctionnement (ex. : modification du flux d'exécution par débordement de tampon).

2. Le code du logiciel ne doit pas pouvoir être modifié en cours d'exécution (ex. : injection de code)

A.2.12.2 ***Séparation des processus***

1. Les processus traitant des informations rouges doivent être isolés.
2. Les processus assurant des fonctions critiques doivent être isolés.
3. Des méthodes contrôlées doivent être utilisées pour transférer des données entre processus.
4. La séparation des processus doit se faire de manière logicielle et matérielle.
5. Le mécanisme de séparation des processus doit être protégé de toute modification (code et structures de données).

A.2.12.3 ***Réutilisation d'objets***

1. Toutes les autorisations d'un objet doivent être révoquées lorsqu'il n'est plus utilisé.
2. Le CSS doit effacer la mémoire qui a été utilisée par un objet avant de l'utiliser pour un autre objet.

Annexe B

Implantation du cahier des charges du CSS

Cette annexe explicite les composants et mécanismes internes du CSS mis en jeu dans l'implantation de chacun des points du cahier des charges défini dans l'annexe précédente. Chaque ligne correspond à un point du cahier des charges, les numéros de ligne correspondent aux numéros de section de chaque point.

		ContB					CommB			I/O				Hw isolation			
		CPU	Firmware	Key register	Crypto accelerator	Secure ICAP	CPU	Firmware	DRCA	Control I/O	Fill I/O	Storage I/O	CommB I/O	Logical isolation (arch)	Electrical isolation	FPGA	SCA framework
Contraintes générales (page 142)	4.8A.2.1.1		x				x	x	x				x				
	4.8A.2.1.2													x	x	x	
	4.8A.2.1.3																
	4.8A.2.1.4					x	x	x	x								
	4.8A.2.1.5				x				x								
initialisation fonctionnement arrêt (page 142)	4.8A.2.2.1.1															x	
	4.8A.2.2.1.2																
	4.8A.2.2.1.3		x		x	x		x	x								
	4.8A.2.2.1.4																x
	A.2.2.2.1		x														x
	A.2.2.2.2		x		x	x											

		ContB					CommB			I/O				Hw isolation		FPGA	SCA framework
		CPU	Firmware	Key register	Crypto accelerator	Secure ICAP	CPU	Firmware	DRCA	Control I/O	Fill I/O	Storage I/O	CommB I/O	Logical isolation (arch)	Electrical isolation		
	A.2.2.2.3							x	x								
	A.2.2.2.4		x														x
	A.2.2.2.5		x	x													
	A.2.2.2.6		x														
	A.2.2.2.7				x												
	A.2.2.3.1		x		x	x		x	x								
	A.2.2.3.2		x			x		x									
	A.2.2.4.1		x					x									
	A.2.2.4.2		x	x				x									
	A.2.2.4.3		x			x		x									
	A.2.2.4.4		x							x							
	A.2.2.5.1		x					x		x	x	x	x				
	A.2.2.5.2		x							x							
	A.2.2.5.3		x							x							
	A.2.2.5.4		x														
	A.2.2.5.5		x		x			x									
Confidentialité (page 143)	A.2.3.1.1							x	x				x	x	x		
	A.2.3.1.2							x	x			x	x	x	x		
	A.2.3.1.3			x				x	x			x	x	x	x		
	A.2.3.1.4					x		x	x								
	A.2.3.1.5							x	x				x				
	A.2.3.1.6							x	x				x				
	A.2.3.2.1							x	x				x	x	x		
	A.2.3.2.2							x	x			x	x	x	x		
	A.2.3.2.3			x				x	x			x	x	x	x		

		ContB					CommB			I/O				Hw isolation		FPGA	SCA framework
		CPU	Firmware	Key register	Crypto accelerator	Secure ICAP	CPU	Firmware	DRCA	Control I/O	Fill I/O	Storage I/O	CommB I/O	Logical isolation (arch)	Electrical isolation		
	A.2.3.2.4					x		x	x								
	A.2.3.2.5							x	x				x				
	A.2.3.2.6							x	x				x				
identification authentification (page 144)	A.2.4.1		x		x			x	x								
	A.2.4.2		x					x		x			x				
	A.2.4.3		x		x			x	x								
	A.2.4.4		x	x								x					
Intégrité (page 144)	A.2.5.1		x		x			x	x								
	A.2.5.2		x					x		x			x				
	A.2.5.3				x				x								
	A.2.5.4		x					x		x			x				
Gestion des clés (page 144)	A.2.6.1.1		x	x				x									
	A.2.6.1.2		x	x							x						
	A.2.6.1.3		x	x													
	A.2.6.1.4		x	x													
	A.2.6.2.1		x									x					
	A.2.6.2.2		x	x	x			x	x								
	A.2.6.2.3		x														
	A.2.6.3.1		x														
	A.2.6.3.2		x		x												
	A.2.6.3.3		x														
	A.2.6.3.4		x	x						x		x	x				
	A.2.6.3.5			x													
	A.2.6.4.1		x	x													
	A.2.6.4.2		x	x													

		ContB					CommB			I/O				Hw isolation		FPGA	SCA framework
		CPU	Firmware	Key register	Crypto accelerator	Secure ICAP	CPU	Firmware	DRCA	Control I/O	Fill I/O	Storage I/O	CommB I/O	Logical isolation (arch)	Electrical isolation		
	A.2.6.4.3		x														
	A.2.6.4.4		x	x								x					
	A.2.6.5.1		x	x													
	A.2.6.5.2		x														
	A.2.6.5.3		x							x							
Gestion des algorithmes (page 145)	A.2.7.1.1		x		x												
	A.2.7.1.2		x		x												
	A.2.7.1.3		x														
	A.2.7.1.4		x														
	A.2.7.1.5		x							x	x						
	A.2.7.2.1		x									x					
	A.2.7.2.2		x									x					
	A.2.7.3.1		x														x
	A.2.7.3.2							x	x								
	A.2.7.3.3		x			x											
	A.2.7.3.4		x			x											
	A.2.7.3.5					x											
	A.2.7.4.1		x			x						x					
Gestion des polices de sécurité (page 146)	A.2.8.1																
	A.2.8.2		x														
	A.2.8.3		x														
	A.2.8.4		x														
	A.2.8.5		x														x
	A.2.9.1							x	x								
	A.2.9.2							x	x								

		ContB					CommB			I/O				Hw isolation		FPGA	SCA framework
		CPU	Firmware	Key register	Crypto accelerator	Secure ICAP	CPU	Firmware	DRCA	Control I/O	Fill I/O	Storage I/O	CommB I/O	Logical isolation (arch)	Electrical isolation		
	A.2.9.3							x	x								
	A.2.9.4		x			x		x									
Module de bypass (page 147)	A.2.10.1.1						x	x									
	A.2.10.1.2							x									
	A.2.10.1.3																
	A.2.10.1.4							x									
	A.2.10.1.5						x	x					x				
	A.2.10.1.6							x									
	A.2.10.1.8		x														
	A.2.10.1.9																
	A.2.10.2.1						x	x									
	A.2.10.2.2		x														x
	A.2.10.2.3						x	x									
Contrôle du CSS (page 147)	A.2.11.1		x		x					x							
	A.2.11.2		x														x
	A.2.11.3																
	A.2.11.4		x							x							
	A.2.11.5		x							x							
	A.2.11.6		x														
Processus (page 147)	A.2.12.1.1	x	x				x	x									
	A.2.12.1.2	x	x				x	x									
	A.2.12.2.1	x	x				x	x									
	A.2.12.1.2	x	x				x	x									
	A.2.12.1.3	x	x				x	x									
	A.2.12.1.4	x	x				x	x									

		ContB					CommB			I/O				Hw isolation		FPGA	SCA framework
		CPU	Firmware	Key register	Crypto accelerator	Secure ICAP	CPU	Firmware	DRCA	Control I/O	Fill I/O	Storage I/O	CommB I/O	Logical isolation (arch)	Electrical isolation		
	A.2.12.1.5	x	x				x	x									
	A.2.12.3.1		x					x									
	A.2.12.3.2	x	x	x	x	x	x	x	x								

Annexe C

Modes d'opération pour les algorithmes de chiffrement par blocs

C.1 L'algorithme de chiffrement par blocs AES

En remplacement du vieillissant DES, le NIST choisit en 2000 l'algorithme Rijndael comme nouveau *standard avancé de chiffrement* (*Advanced Encryption Standard AES*). L'algorithme AES [Nist01a] est un algorithme de chiffrement symétrique par bloc de 128 bits qui permet l'utilisation de clés de 128, 192 ou 256 bits. Selon la taille de la clé utilisée, AES est composé de 10, 12 ou 14 rounds presque identiques. Chaque round est composé de diverses transformations linéaires et non-linéaire telles que des permutations, des XOR ou encore des multiplications et inversions dans des corps de Galois.

Utilisé seul, l'algorithme AES laisse filtrer un certain nombre d'informations au sujet de la structure du texte clair qu'il sert à chiffrer ce qui fait que ce mode d'opération appelé *Electronic Code Book* (ECB) n'est pas sûr. Pour remédier à ce problème, plusieurs modes d'opération ayant un meilleur niveau de sécurité ont été conçus. La suite de cet annexe en présente cinq : les modes CTR, CBC et CBC-MAC et les modes CCM et GCM qui sont basés sur les trois précédents.

C.2 Mode de chiffrement

Le mode d'opération le plus simple après ECB est le mode CBC (*Chained Block Cipheri*) [Nist01b]. Ce dernier permet de casser les structures répétitives qui peuvent apparaître lorsque le mode de chiffrement ECB est utilisé. Avec le mode CBC, chaque bloc de texte clair est préalablement additionné (XOR) au bloc de texte chiffré précédent avant d'être chiffré à son tour. Pour le premier bloc de texte clair, un *vecteur d'initialisation* (IV) est utilisé. La Figure 50 illustre le fonctionnement de ce mode d'opération lors d'u chiffrement d'un texte clair. Afin de préserver la sécurité de ce mode d'opération, l'utilisateur doit veiller à ce que le même vecteur d'initialisation ne soit jamais utilisé deux fois avec une même clé.

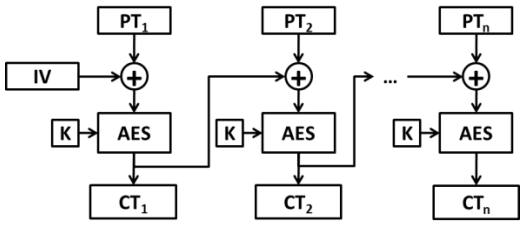


Figure 50 : Mode d'opération CBC

Initialement utilisé pour le chiffrement de données, le mode CBC peut aussi être utilisé pour leur authentification. En effet, avec CBC, la valeur de chaque bloc de texte chiffré dépend de l'ensemble des précédentes valeurs des blocs de texte clair. Il s'ensuit que le dernier bloc de texte chiffré d'un message peut être utilisé comme TAG d'authentification dudit message. Ce mode d'opération porte le nom de CBC-MAC. Il est à noter que lorsque ce mode est utilisé, il n'est pas possible d'utiliser CBC pour chiffrer le message. L'utilisation de CBC exclue automatiquement l'utilisation de CBC-MAC et réciproquement.

Le principal inconvénient du mode CBC réside dans la nécessité d'avoir terminé le chiffrement du bloc de texte clair précédent avant de pouvoir commencer le chiffrement du bloc de texte courant. Le mode de chiffrement *Counter* (CTR) a été développé dans le but d'apporter une solution à ce problème. Avec ce mode de chiffrement, un compteur est incrémenté avant le chiffrement d'un bloc de texte clair. C'est la valeur de ce compteur qui par la suite est chiffrée avant de l'additionner (XOR) avec la valeur du bloc de texte clair afin d'obtenir le bloc de texte chiffré. La Figure 51 illustre le fonctionnement du mode CTR. Afin de garantir la sécurité de ce mode de chiffrement, une même valeur de compteur ne doit jamais être utilisée deux fois avec la même clé.

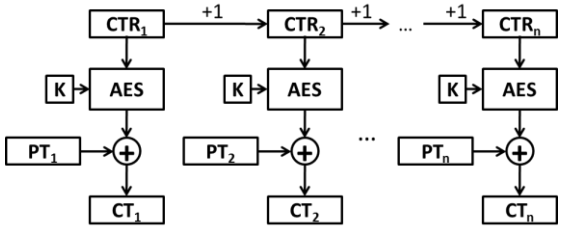


Figure 51 : Mode d'opération CTR

C.3 **Mode de chiffrement avec authentification**

A l'exception de CBC-MAC qui ne permet que l'authentification d'un texte clair, les modes précédents ne permettent que le chiffrement d'un texte clair. Or, il est en général nécessaire de procéder à la fois au chiffrement et à l'authentification d'un message. Les deux modes présentés dans cette partie permettent de faire cela.

Le mode CCM (*Counter with CBC-MAC*) [WhHF02] met en œuvre les deux modes précédent afin de fournir un service de chiffrement authentifié. CCM fonctionne e la façon suivante : tout d'abord l'en-tête de message est calculée comme le spécifie la définition du mode puis, l'en-tête, les données à authentifier et les données à authentifier et chiffrer sont authentifiées à l'aide de CBC-MAC. Pour finir, les données à chiffrer et le condensé calculé par CBC-MAC sont chiffrés. On notera que les opérations d'authentification et de chiffrement peuvent se faire en parallèle mais que le débit de CBC-MAC est limité par la rapidité du processus d'authentification. La Figure 52 illustre ce mode de fonctionnement.

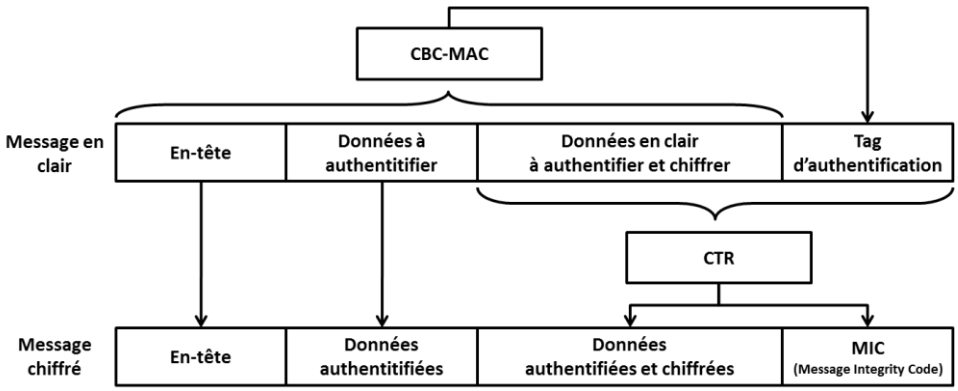


Figure 52 : Mode d'opération CCM

Pour pallier aux faibles performances du mode CCM, le mode GCM (*Galois Counter Mode*) a été conçu. Tout comme CCM, le mode GCM utilise le mode CTR mais, pour l'authentification, il a recourt à une multiplication dans un corps de Galois. Cette multiplication bien plus rapide qu'une opération de chiffrement AES permet de concevoir des implantations déroulées et pipelinées de GCM dont le débit en sortie est d'un bloc de texte chiffré par cycle d'horloge. La Figure 53 illustre le fonctionnement du mode GCM, la valeur H est égale au chiffrement par AES de la valeur 0 avec la clé de chiffrement Ek.

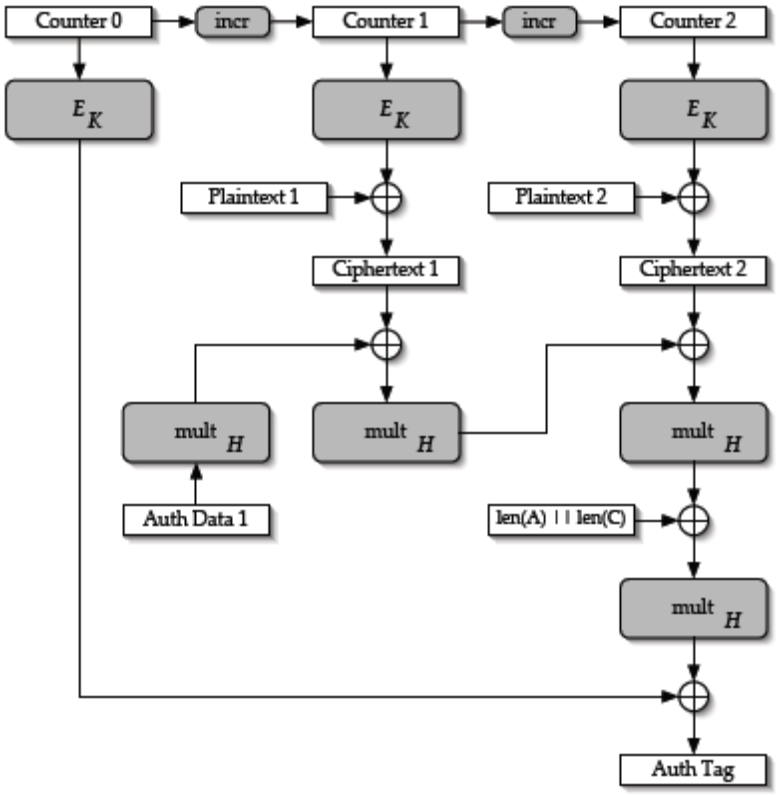


Figure 53 : Mode de chiffrement authentifié GCM

Publications

- Grand M., Bossuet L., Gogniat G., Le Gal B., Dallet D., « A Reconfigurable Crypto Sub System for the Software Communication Architecture », Military Communications Conference, 2009. MILCOM 2009. IEEE, Boston, USA.

- Grand M., Bossuet L., Gogniat G., Le Gal B., Dallet D., « A multi-core AES cryptoprocessor for multi-channel SDR », Military Communiacations and Information Systems Technology Week, MCISWEEK 2010, Wroclaw, Pologne.

- Le Gal B., Bossuet L., Grand M., *Enseignement ludique de la programmation objets à l'aide des applications de traitement d'image*. Journal sur l'enseignement des sciences et technologies de l'information et des systèmes (j3ea), EDP Sciences, vol.10, juin 2011. http://www.j3ea.org/10.1051/j3ea/2010012.

- Grand M., Bossuet L., Le Gal B., Gogniat G., Dallet D., « Design and Implementation of a Multi-Core Crypto-Processor for Software Defined Radio »,;In proceedings of the 7th international symposium on applied reconfigurable computing, ARC 2011, Belfast, Royaume-Uni.

- Grand M., Bossuet L., Le Gal B., Gogniat G., Delahaye J.P., Dallet D., « A Reconfigurable Multi-core Cryptoprocessor for Multi-channel Communication Systems », In proceedeings of the 18th Reconfigurable Architecture Workshop, RAW 2011, Anchorage, USA.

- Bossuet L., Grand M., Gaspard L., Gogniat G., Fischer V., Dallet D., « Architectures of flexible symmetric key crypto-engines – a survey: from hardware coprocessor to multi-crypto-processor system on chip » *article soumis à la revue ACM Computing Surveys (2011)*

Bibliographie

[ARRS06] D. Arora, A. Raghunathan, S. Ravi, M. Sankaradass, N. K. Jha, and S. T. Chakradhar, "Software architecture exploration for high-performance security processing on a multiprocessor mobile SoC," in *In proceedings of the 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 496-501.

[Alte09] Altera, *AN567: Quartus II Design Separation Flow*. 2009.

[AzIk07] A. Aziz and N. Ikram, "An FPGA-based AES-CCM crypto core for IEEE 802.11 i architecture," *International Journal of Networks Security*, vol. 5, no. 2, pp. 224-232, 2007.

[BCNT06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370-382, 2006.

[BaHK06] M. Barbeau, J. Hall, and E. Kranakis, "Detecting Impersonation Attacks in Future Wireless and Mobile Networks," in *Secure Mobile Ad-hoc Networks and Sensors*, vol. 4074, M. Burmester and A. Yasinsac, Eds. Springer Berlin / Heidelberg, 2006, pp. 80-95.

[BaMV05] L. Batina, N. Mentens, and I. Verbauwhede, "Side-channel issues for designing secure hardware implementations," in *11th IEEE International On-Line Testing Symposium*, 2005, pp. 118-121.

[BeMS08] I. Beretta, G. Mangano, and M. D. Santambrogio, *How to provide a Linux support for dynamic reconfiguration on Xilinx FPGAs*. Milan: , 2008.

[BeSa03] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: real vulnerabilities and practical solutions," in *Proceedings of the 12th conference on USENIX Security Symposium*, 2003, p. 2.

[BoAn01] M. Bond and R. Anderson, "API-level attacks on embedded systems," *Computer*, vol. 34, no. 10, pp. 67-75, 2001.

[BoGP05] L. Bossuet, G. Gogniat, and J.-L. Philippe, "Generic Design Space Exploration for Reconfigurable Architectures," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, p. 163a-163a.

[BoMK08] B. C. Boorman, C. D. Mackey, and M. T. Kurdziel, "A scalable hardware architecture to support applications of the haipe 3.1 standard," in *Military Communications Conference, 2007. MILCOM 2007. IEEE*, 2008, pp. 1–8.

[BuHO04] R. Buchty, N. Heintze, and D. Oliva, "Cryptonite–A programmable crypto processor architecture for high-bandwidth applications," in *Organic and Pervasive Computing–ARCS 2004*, 2004, vol. 2981/2004, pp. 184–198.

[BuMA00] J. Burke, J. McDonald, and T. Austin, "Architectural support for fast symmetric-key cryptography," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 5, pp. 178-189, Dec. 2000.

[CCCC05] S. Chih-Pin, H. Chia-Lung, H. Chih-Tsun, and W. Cheng-Wen, "A configurable AES processor for enhanced security," in *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, 2005, pp. 361-366.

[CHLM06] J. Castillo, P. Huerta, V. Lopez, and J. I. Mart'\inez, "A secure self-reconfiguring architecture based on open-source hardware," in *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on*, 2006, vol. 0, pp. 7–10.

[CKVS06] R. Chaves, G. Kuzmanov, S. Vassiliadis, and L. Sousa, "Reconfigurable Cryptographic Processor," in *In proceeding of the Workshop on Circuits, Systems and Signal Processing*, 2006.

[CZSB08] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, *A multi-platform controller allowing for maximum Dynamic Partial Reconfiguration throughput*. IEEE, 2008, pp. 535-538.

[ChGa00] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementations of the AES Algorithm Why Yet Another AES Implementation ?"

[ChGa03] P. Chodowiec and K. Gaj, "Very Compact FPGA Implementation of the AES Algorithm," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, 2003, vol. 2779, pp. 319-333.

[Clos53] C. Clos, "A Study of Non-Blocking Switching Networks," *Bell System Technical Journal*, vol. 32, no. 5, pp. 406-424, 1953.

[CoBo99] P. G. Cook and W. Bonser, "Architectural overview of the SPEAKeasy system," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, pp. 650-661, Apr. 1999.

[CuRe95] Y. Cui-Qing and A. V. S. Reddy, "A taxonomy for congestion control algorithms in packet switching networks," *YIEEE Network*, vol. 9, no. 4, pp. 34-45, 1995.

[EYCP01] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 4, pp. 545-557, 2001.

[ElPa03] A. J. Elbirt and C. Paar, "Instruction-level distributed processing for symmetric-key cryptography," in *Proceedings International Parallel and Distributed Processing Symposium*, 2003, p. 10.

[Ettu11] Ettus Research LLC, "USRP™ Family Products and Daughter Boards," 2011. [Online]. Available: http://www.ettus.com/products.

[FePC05] A. Ferrante, V. Piuri, and F. Castanier, *A QdS-enabled packet scheduling algorithm for IPSec multi-accelerator based systems*. New York, New York, USA: ACM Press, 2005, p. 221.

[FlMS01] S. R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," in *In proceedings of the 8th Annual International Workshop on Selected Areas in Cryptography*, 2001, pp. 1-24.

[FrPP08] D. Fronte, A. Perez, and E. Payrat, "Celator: A Multi-algorithm Cryptographic Co-processor," in *Proc. International Conference on Reconfigurable Computing and FPGAs ReConFig '08*, 2008, pp. 438-443.

[GCSB06] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An Overview of Reconfigurable Hardware in Embedded Systems," *EURASIP Journal on Embedded Systems*, vol. 2006, no. 1, pp. 1-19, Jan. 2006.

[GDSV09] C. R. A. González et al., "Open-source SCA-based core framework and rapid development tools enable software-defined radio education and research," *Communications Magazine, IEEE*, vol. 47, no. 10, pp. 48–55, 2009.

[GFBB10] L. Gaspar, V. Fischer, F. Bernard, L. Bossuet, and P. Cotret, "HCrypt: A Novel Concept of Crypto-processor with Secured Key Management," in *2010 International Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 280-285.

[GKAR10] K. Gaj, J.-P. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster, *ATHENa - Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs*. IEEE, 2010, pp. 414-421.

[GMND11] S. Guilley et al., "Vade Mecum on Side-Channels Attacks and Countermeasures for the Designer and the Evaluator," in *In proceedings of the 6th International conference on Design & Technology of Integrated Systems in nanoscale aera*, 2011.

[GaFB11] L. Gaspard, V. Fisher, and L. Bossuet, "Secure extension of softcore genral-purpose processors for symmetric key cryptography," in *6th international Workshop on Reconfigurable Communication Centric Systems on Chip, ReCoSoc 2011*, 2011.

[Gene06] General Dynamics, "Advanced INFOSEC Machine Datasheet." General Dynamics, 2006.

[Gila08] R. Giladi, *Network Processors*, Systems on. Morgan Kaufman, 2008, p. 722.

[GoWB06] G. Gogniat, T. Wolf, and W. Burleson, "Reconfigurable Security Support for Embedded Systems," in *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, 2006, vol. 10, p. 250a-250a.

[Goub01] L. Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," in *Cryptographic Hardware and Embedded Systems — CHES 2001*, vol. 2162, Ç. Koç, D. Naccache, and C. Paar, Eds. Springer Berlin / Heidelberg, 2001, pp. 3-15.

[GrBG10] M. Grand, L. Bossuet, and B. L. Gal, "A reconfigurable crypto sub system for the software communication architecture," in *Military Communications Conference, 2009. MILCOM 2009. IEEE*, 2010.

[GuCh08] H. Gu and S. Chen, *Partial Reconfiguration Bitstream Compression for Virtex FPGAs*. IEEE, 2008, pp. 183-185.

[Guer10] S. Gueron, "Intel Advanced Encryption Standard (AES) Instructions Set." Intel Mobility Group, pp. 1-79, 2010.

[HBWS07] T. Huffmire et al., "Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007, pp. 281-295.

[HUWB04] M. Huebner, M. Ullmann, F. Weissel, and J. Becker, "Real-time configuration code decompression for dynamic FPGA self-reconfiguration," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 138-143.

[Harr05] Harris Corp., "Sierra II Datasheet." Harris Corp., 2005.

[HeCW08] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Aerospace Conference, 2008 IEEE*, 2008, pp. 1–10.

[HoVe04] A. Hodjat and I. Verbauwhede, "Interfacing a high speed crypto accelerator to an embedded CPU," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, 2004, pp. 488-492.

[HoVe06] A. Hodjat and I. Verbauwhede, "Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors," *IEEE Transactions on Computers*, vol. 55, pp. 366–372, 2006.

[Hui95] Z. Hui, "Service disciplines for guaranteed performance service in packet-switching networks," *Proceedings of the IEEE*, vol. 83, no. 10, pp. 1374-1396, 1995.

[HäHH07] P. Hämäläinen, M. Hännikäinen, and T. Hämäläinen, "Review of Hardware Architectures for Advanced Encryption Standard Implementations Considering Wireless Sensor Networks," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, vol. 4599, S. Vassiliadis, M. Berekovic, and T. Hämäläinen, Eds. Springer Berlin / Heidelberg, 2007, pp. 443-453.

[Itu00a] ITU, "Specifications G.711." [Online]. Available: http://www.itu.int/rec/T-REC-G.711/fr.

[Itu00b] ITU, "Specification G.729." [Online]. Available: http://www.itu.int/rec/T-REC-G.729/fr.

[JSMY09] A. Joux, F.-X. Standaert, T. Malkin, and M. Yung, "Advances in Cryptology - EUROCRYPT 2009," 2009, vol. 5479, pp. 443-461-461.

[JXJB09] G. Jianxin, Y. Xiaohui, G. Jun, and W. Bo, "The flow of software defined radio waveform development based on SCARI," in *5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09.*, 2009, pp. 1–4.

[JiKW04] M. H. Jing, S. Y. Ko, and W. C. Wu, "The SOC design of a highly secure and reliable storage using a conceptual environment," in *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems, 2004. Proceedings.*, 2004, pp. 865-868.

[Jtrs04] JTRS, "Security Supplement to the Software Communications Architecture Specification," no. 2.2.1. JTRS, 2004.

[Jtrs06] JTRS, "Software Communications Architecture Specification," no. 2.2.2. JTRS, May-2006.

[Jtrs07] JTRS, "Modem Hardware Abstraction Layer Application Program Interface," no. 2.11.1. JTRS, 2007.

[Jtrs10] JTRS, "Software communications architecture specification, Next Version," no. Next. 2010.

[KMKS08] K. Kepa, F. Morgan, K. Kosciuszkiewicz, and T. Surmacz, *SeReCon: A Secure Dynamic Partial Reconfiguration Controller*. IEEE, 2008, pp. 292-297.

[KaBG08] N. Kamoun, L. Bossuet, and A. Ghazel, *SRAM-FPGA implementation of masked S-Box based DPA countermeasure for AES*. IEEE, 2008, pp. 74-77.

[KaVE95] M. Katevenis, P. Vatsolaki, and A. Efthymiou, "Pipelined memory shared buffer for VLSI switches," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 4, pp. 39-48, Oct. 1995.

[Ken00] C. Ken, "PicoBlaze User Resources." Xilinx.

[Kerc83] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, vol. IX, pp. 5-38, 1883.

[KuBF05] M. Kurdziel, J. Beane, and J. J. Fitton, "An SCA security supplement compliant radio architecture," in *Proc. IEEE Military Communications Conference MILCOM 2005*, 2005, pp. 2244-2250.

[KuWF05] D. R. Kuhn, T. J. Walsh, and S. Fries, *Security Considerations for Voice Over IP Systems*. Gaithersburg: NIST, 2005.

[LAMT07] R. Leveugle et al., "Experimental evaluation of protections against laser-induced faults and consequences on fault modeling," in *Proceedings of the conference on Design, automation and test in Europe DATE '07*, 2007, pp. 1587-1592.

[LKLJ09a] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, *Run-time Partial Reconfiguration speed investigation and architectural design space exploration*. IEEE, 2009, pp. 498-502.

[LKLJ09b] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time Partial Reconfiguration speed investigation and architectural design space exploration," in *Proc. Int. Conf. Field Programmable Logic and Applications FPL 2009*, 2009, pp. 498-502.

[LKLJ09c] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, *Run-time Partial Reconfiguration speed investigation and architectural design space exploration*. IEEE, 2009, pp. 498-502.

[LWFB07] S. Lemsitzer, J. Wolkerstorfer, N. Felber, and M. Braendli, "Multi-gigabit GCM-AES Architecture Optimized for FPGAs," in *CHES '07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, 2007, pp. 227-238.

[LeKS10] J. Lee, K. Kapitanova, and S. H. Son, "The price of security in wireless sensor networks," *Computer Networks*, vol. 54, no. 17, pp. 2967-2978, Dec. 2010.

[LoRD06] E. Lopez-Trejo, F. Rodriguez-Henriquez, and A. Diaz-Pérez, "An FPGA Implementation of CCM Mode Using AES," *Information Security and Cryptology-ICISC 2005*, pp. 322–334, 2006.

[Lomo04] M. Lomonaco, "Cryptarray a Scalable and Reconfigurable Architecture for Cryptographic Applications," University of Central Florida, USA, 2004.

[MTRG99] E. Mosanya, C. Teuscher, H. Restrepo, P. Galley, and E. Sanchez, "CryptoBooster: A Reconfigurable and Modular Cryptographic Coprocessor," in *Cryptographic Hardware and Embedded Systems*, vol. 1717, Ç. Koç and C. Paar, Eds. Springer Berlin / Heidelberg, 1999, p. 726.

[MVCT07] C. Mucci, L. Vanzolini, F. Campi, and M. Toma, "Interactive presentation: Implementation of AES/Rijndael on a dynamically reconfigurable architecture," in *Proceedings of the conference on Design, automation and test in Europe (DATE)*, 2007, pp. 355-360.

[MaNM08] A. Martin, T. R. Newman, and D. Murotake, "Development Approaches for an International Tactical Radio Cryptographic API," in *Proceedings of the SDR'08 Technical Conference and Product Exposition*, 2008.

[McMM07] M. McLean, J. Moore, and F. Meade, "FPGA-based single chip cryptographic solution," *Military Embedded Systems*, 2007.

[MeWe04] U. Meyer and S. Wetzel, "A man-in-the-middle attack on UMTS," in *Proceedings of the 2004 ACM workshop on Wireless security - WiSe '04*, 2004, p. 90.

[MiRe04] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, p. 39, Apr. 2004.

[Mito95] J. Mitola, "The software radio architecture," *IEEE Communications Magazine*, vol. 33, no. 5, pp. 26-38, May. 1995.

[Mok83] A. K. Mok, "FUNDAMENTAL DESIGN PROBLEMS OF DISTRIBUTED SYSTEMS FOR THE HARD-REAL-TIME ENVIRONMENT." Massachusetts Institute of Technology, 1983.

[MuMa06] D. Murotake and A. Martin, "A HIGH ASSURANCE WIRELESS COMPUTING SYSTEM (HAWCS) FOR SOFTWARE DEFINED RADIO," in *Proceeding of the SDR 06 Technical Conference and Product Expositi*, 2006.

[NaSa81] D. Nassimi and S. Sahni, "A Self-Routing Benes Network and Parallel Permutation Algorithms," *IEEE Transactions on Computers*, vol. 30, no. 5, pp. 332-340, May. 1981.

[Nist01a] NIST, "FIPS-197." Nist, 2001.

[Nist01b] NIST, "Special Publication 800-38A." Nist, 2001.

[Nist04] NIST, "Special Publication 800-38C." Nist, 2004.

[Nist07] NIST, "Special Publication 800-38D." NIST, 2007.

[PCGV08] M. Pericàs, R. Chaves, G. Gaydadjiev, S. Vassiliadis, and M. Valero, "Vectorized AES Core for High-throughput Secure Environments," in *High Performance Computing for Computational Science - VECPAR 2008*, vol. 5336, J. Palma, P. Amestoy, M. Daydé, M. Mattoso, and J. Lopes, Eds. Springer Berlin / Heidelberg, 2008, pp. 83-94.

[Perl93] C. Perleberg, "Branch target buffer design and optimization," *IEEE Transactions on Computers*, vol. 42, no. 4, pp. 396-412, 1993.

[PiMS91] R. L. Pickholtz, L. B. Milstein, and D. L. Schilling, "Spread spectrum for mobile communications," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 2, pp. 313-322, May. 1991.

[RDBD08] F. Rivet, Y. Deval, J.-B. Bégueret, D. Dallet, P. Cathelin, and D. Belot, "A Disruptive Receiver Architecture Dedicated to Software-Defined Radio," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 4, pp. 344-348, 2008.

[RRPS02] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass, "System design methodologies for a wireless security processing platform," in *Proceedings of the 39th conference on Design automation - DAC '02*, 2002, p. 777.

[RaPa03] S. Ramabhadran and J. Pasquale, "Stratified round Robin: a low complexity packet scheduler with bandwidth fairness and bounded delay," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, 2003, p. 239.

[RaTa10] P. Radmand and A. Talevski, "Impact of Encryption on QdS in Voip," in *2010 IEEE Second International Conference on Social Computing*, 2010, pp. 721-726.

[Rayt00] Raytheon, "Cornfield Multi-Chip Module." [Online]. Available: http://www.fas.org/irp/program/security/_work/cornfld.html.

[RhMi90] U.-S. Rhee and M. M. Mirsalehi, "Two-dimensional Benes network," in *Proceedings. The Twenty-Second Southeastern Symposium on System Theory*, 1990, pp. 614-619.

[SACA08] S. Singh, M. Adrat, S. Couturierand, M. Antweiler, M. Phisel, and S. Bernier, "SCA BASED IMPLEMENTATION OF STANAG 4285 IN A JOINT EFFORT UNDER THE NATO RTO/IST PANEL," in *Proceedings of the SDR'08 Technical Conference and product Exposition*, 2008.

[SBPV06] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Superscalar coprocessor for high-speed curve-based cryptography," 2006, pp. 415-429.

[ScVe03] P. Schaumont and I. Verbauwhede, "Domain-specific codesign for embedded security," *Computer*, vol. 36, no. 4, pp. 68-74, Apr. 2003.

[ShUA00] H. Shiba, K. Uehara, and K. Araki, "Proposal and evaluation of security schemes for software-defined radio," in *14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications, 2003. PIMRC 2003.*, pp. 114-118.

[SiSe10] S. Sidharth and M. P. Sebastian, "A Revised Secure Authentication Protocol for IEEE 802.16 (e)," in *2010 International Conference on Advances in Computer Engineering*, 2010, pp. 34-38.

[Skor09] S. Skorobogatov, "Local heating attacks on Flash memory devices." IEEE, Jul-2009.

[StIR04] A. Stubblefield, J. Ioannidis, and A. D. Rubin, "A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP)," *ACM Transactions on Information and System Security*, vol. 7, no. 2, pp. 319-332, May. 2004.

[Stan10] F.-X. Standaert, "Introduction to Side-Channel Attacks," in *Secure Integrated Circuits and Systems*, I. M. R. Verbauwhede, Ed. Springer US, 2010, pp. 27-42.

[SuGA07] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007.

[SuMy05] R. H. Suvda and R. L. H. S. Myagmar, "Threat Analysis of GNU Software Radio." 2005.

[SzHa04] T. Szigeti and C. Hattingh, *End-to-End QdS Network Design: Quality of Service in LANs, WANs, and VPNs (Networking Technology)*. Cisco Press, 2004.

[Tcpa03] TCPA – TRUSTED COMPUTING PLATFORM ALLIANCE, "TPM Main Specification Version 1.1b." Trusted Computing Group, 2003.

[TeBe09] E. Tews and M. Beck, *Practical attacks against WEP and WPA*. New York, New York, USA: ACM Press, 2009, p. 79.

[ThPP08] D. Theodoropoulos, I. Papaefstathiou, and D. Pnevmatikatos, "CCproc: An Efficient Cryptographic Coprocessor," in *In proceedings of 16th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI 2008)*, 2008, pp. 160-163.

[ThSP09] D. Theodoropoulos, A. Siskos, and D. Pnevmatikatos, "CCproc: A Custom VLIW Cryptography Co-processor for Symmetric-Key Ciphers," in *Reconfigurable Computing: Architectures, Tools and Applications*, vol. 5453, J. Becker, R. Woods, P. Athanas, and F. Morgan, Eds. Springer Berlin / Heidelberg, 2009, pp. 318-323.

[TiGS00] S. TILLICH, J. GROSSSCHÄDL, and A. SZEKELY, "An instruction set extension for fast and memory-efficient AES implementation," *Lecture notes in computer science*, pp. 11-21.

[TiGr06] S. Tillich and J. Großschädl, "Instruction Set Extensions for Efficient AES Implementation on 32-bit Processors," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, vol. 4249, L. Goubin and M. Matsui, Eds. Springer Berlin / Heidelberg, 2006, pp. 270-284.

[TiHe08] S. Tillich and C. Herbst, "Boosting AES Performance on a Tiny Processor Core," in *Topics in Cryptology – CT-RSA 2008*, vol. 4964, T. Malkin, Ed. Springer Berlin / Heidelberg, 2008, pp. 170-186.

[TrKo10] E. Trichina and R. Korkikyan, *Multi Fault Laser Attacks on Protected CRT-RSA*. IEEE, 2010, pp. 75-86.

[TrSh03] N. Tredennick and B. Shimamoto, "The Rise of Reconfigurable Systems," in *proceedings of Engineering of Reconfigurable Systems and Application conference*, 2003.

[Turn05] M. R. Turner, "SOFTWARE DEFINED RADIO SOLUTIONS Experience making JTRS work, from the SCA, to Waveforms, to Secure Radios," in *Proceeding of the SDR 05 Technical Conference and Product Exposition*, 2005.

[UcUK00] H. Uchikawa, K. Umebayashi, and R. Kohn, *Secure download system based on software defined radio composed of FPGAs*. IEEE, pp. 437-441.

[VWGB04] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN polymorphic processor," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363-1375, Nov. 2004.

[WKWA01] C. Weaver, R. Krishna, L. Wu, and T. Austin, "Application specific architectures: a recipe for fast, flexible and power efficient designs," in *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems - CASES '01*, 2001, p. 181.

[WSHW10] M.-Y. Wang, C.-P. Su, C.-L. Horng, C.-W. Wu, and C.-T. Huang, "Single- and Multi-core Configurable AES Architectures for Flexible Security," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 4, pp. 541-552, Apr. 2010.

[Wein00] S. Weingart, "Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses," *Cryptographic Hardware and Embedded Systems — CHES 2000*, vol. 1965. Springer Berlin / Heidelberg, pp. 45-68, 2000.

[WhHF02] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM), Submission to NIST." 2002.

[WoGP04] T. Wollinger, J. Guajardo, and C. Paar, "Security on FPGAs: State-of-the-art implementations and attacks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 3, pp. 534-574, Aug. 2004.

[WoGa03] K. Wongthavarawat and A. Ganz, "Packet scheduling for QdS support in IEEE 802.16 broadband wireless access systems," *International Journal of Communication Systems*, vol. 16, no. 1, pp. 81-96, Feb. 2003.

[WoPa03] T. Wollinger and C. Paar, "How Secure Are FPGAs in Cryptographic Applications?," in *Field Programmable Logic and Application*, vol. 2778, P. Y. K. Cheung and G. Constantinides, Eds. Springer Berlin / Heidelberg, 2003, pp. 91-100.

[WuWA02] L. Wu, C. Weaver, and T. Austin, "CryptoManiac: a fast flexible architecture for secure communication," in *28th Annual International Symposium on Computer Architecture, 2001*, 2002, pp. 110–119.

[Xili03] Xilinx, "CryptoBlaze: 8-bit Security Microcontroller (XAPP374)." Sep-2003.

[Xili08] Xilinx, "LYRtech Virtex-4 FPGA Software Defined Radio (SDR) Development Platform," 2008. [Online]. Available: http://www.xilinx.com/products/boards-and-kits/SFF-SDR-DP.htm.

[Xili10a] Xilinx, "UG360: Virtex-6 FPGA Configuration." 2010.

[Xili10b] Xilinx, *Fast Simplex Link Bus*. 2010.

[Xili11a] Xilinx, "UG702: Partial Reconfiguration User Guide." Xilinx, 2011.

[Xili11b] Xilinx, *MicroBlaze Processor Reference Guide*. 2011.

[ZSFZ11] C. Zhang, Y. Song, Y. Fang, and Y. Zhang, "On the Price of Security in Large-Scale Wireless Ad Hoc Networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 2, pp. 319-332, Apr. 2011.

[Zein05] A. H. S. Zeineddini, "Secure partial reconfiguration of {FPGAs}," George Mason University, 2005.