Contre-mesures

Les chapitres précédents ont présenté des chemins d'attaques efficaces qui exploitent différents vecteurs d'attaque tels qu'un cheval de Troie matériel, une IP tiers malveillante, un outil CAO corrompu ou un concepteur malveillant. Ces attaques permettent de réaliser une élévation de privilège, un déni de service, un vol de clés cryptographiques ou de données confidentielles.

Ce chapitre présente des solutions pour améliorer la sécurité des SoC FPGA embarquant la technologie ARM TrustZone en se protégeant des attaques présentées précédemment. Il présente une méthodologie d'isolation des IP sécurisées, mais aussi un système de vérification des règles de conception et une étude de l'utilisation d'un chiffrement de flux pour sécuriser les transactions sur le bus AXI.

1. Protection #1 : Isolation des IP sécurisées de la partie reconfigurable du SoC

Cette section présente une méthodologie d'isolation matérielle et logicielle des IP sécurisées de la partie reconfigurable d'un SoC complexe hétérogène embarquant la technologie ARM TrustZone. Cette méthodologie est composée de trois étapes et utilise une technique d'isolation matérielle et des configurations logicielles pour protéger les IP sécurisées et les régions sécurisées de la mémoire externe. La figure 48 donne un aperçu de l'isolation des IP sécurisées (encadrées en violet) dans la partie reconfigurable du SoC.

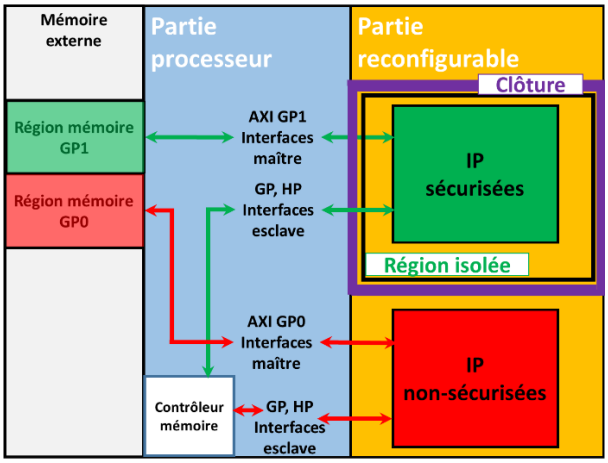

Figure 47: L'isolation des IP sécurisées dans la partie reconfigurable du SoC

Dans cette méthodologie, les IP sécurisées à isoler doivent être des IP de confiance, c'est-à-dire que l'intégrateur doit avoir l'assurance qu'elles ne contiennent pas de vulnérabilité. Des certifications (à l'image de celles proposées pour les Trust Applications que l'on peut exécuter dans une TEE) pourraient être envisagées dans ce but. Cependant, aujourd'hui il n'existe pas de certifications pour cela et c'est avant tout un lien de confiance entre l'intégrateur et le concepteur de l'IP qui doit permettre de réduire les risques (c'est le cas si par exemple l'intégrateur et le concepteur de l'IP font partie de la même société).

Dans le cas d'un SoC du type Xilinx Zynq, la méthode que nous proposons pour l'isolation des IP sécurisées de la partie reconfigurable est constituée des trois étapes suivantes :

Etape 1 : **Isolation physique et logique des IP sécurisées dans la partie reconfigurable du SoC.** Dans cette étape, le concepteur doit utiliser la méthodologie d'isolation du flux de conception (Isolation Design Flow [78], IDF) de Xilinx qui permet à des IP indépendantes de fonctionner sur un seul FPGA avec une séparation logique et physique. L'IDF implémente et route

les IP dans des zones isolées et entourées d'une « clôture » composée d'un ensemble d'éléments logiques configurables non configurés (sans fonction logique) dans lesquels aucun routage n'est présent (pas de connexions internes ni externes, pas de ligne de routage traversant les éléments logiques configurables en dehors des connexions aux entrées-sorties de l'IP isolée). L'IDF utilise également un outil indépendant de vérification des règles de conception (*Design Rule Check*, DRC en anglais), pour vérifier la clôture et les connexions avec la zone isolée.

Etape 2 : Utilisation de deux interfaces maîtres de la partie processeur du SoC. Comme cela est représenté au centre de la figure 48, dans la partie processeur du SoC, deux interfaces vont contrôler les interfaces des IP sécurisées (connexions vertes sur la figure 48) et pour contrôler celles des IP non-sécurisées (connexions rouges sur la figure 48). Dans le cas du SoC Xilinx Zynq-7000, la partie processeur possède deux interfaces maîtres à usage général : l'interface AXI GP0 et l'interface AXI GP1. Ces deux interfaces peuvent être utilisées pour contrôler les interfaces esclaves des IP sécurisées et non sécurisées comme cela est présenté sur la figure 48. Sur cette figure, l'interface AXI GP1 est utilisé pour contrôler les IP sécurisées et l'interface AXI GP0 est utilisé pour contrôler les IP non sécurisées. Chacun de ces deux interfaces a une région de la mémoire externe qui lui est dédiée (la région mémoire GP1 pour l'interface AXI GP1 et la région mémoire GP0 pour l'interface AXI GP0 comme cela apparait sur la figure 48). De plus, le SoC Xilinx Zynq-7000 propose des registres de configuration liés à chacune des deux interfaces maîtres AXI GP1 et AXI GP0. Ces registres configurent la propagation de l'état de sécurité des requêtes de la partie processeur vers la partie reconfigurable du SoC. Par exemple, dans la figure 48, comme il contrôle les IP non sécurisées, l'interface AXI GP0 est configuré pour interdire la propagation de l'état de sécurité des requêtes de la partie processeur vers les IP sécurisées. Toute requête non-sécurisée destinée aux IP sécurisées (accès à la région mémoire GP1 (en vert dans la figure 48) de la mémoire externe dédiée à l'interface AXI GP1) est rejetée au niveau de l'interface AXI GP0, et une exception est alors déclenchée dans la partie processeur du SoC.

Etape 3 : Limitation des accès aux régions sécurisées de la mémoire externe. Pour cette étape, le concepteur doit interdire aux interfaces maîtres des IP non-sécurisées d'accéder aux régions sécurisées de la mémoire externe. Dans le cas du SoC Xilinx Zynq-7000, la partie processeur possède deux interfaces esclaves AXI GP (interface esclave GP0 et interface esclave GP1), quatre interfaces haute performance (HP) et une interface ACP pour la cohérence de la mémoire cache (dont nous avons montré l'exploitation malicieuse dans le chapitre précédant).

Ces interfaces permettent théoriquement aux IP de la partie reconfigurable du SoC un accès direct à l'intégralité de la mémoire externe. Cependant, il est possible de limiter cet accès à des régions spécifiques. Pour cela, les interfaces esclaves AXI GP et HP disponibles doivent être scindées en deux groupes distinct pour être connectés aux interfaces maîtres des IP sécurisées (connexions vertes sur la figure 48) et non-sécurisées (connexions rouges sur la figure 48) de la partie reconfigurable du SoC. Pour limiter la possibilité de réaliser les attaques ciblant la mémoire cache (voir chapitre 4), l'utilisation de l'interface ACP doit être strictement limitée aux seules IP sécurisées de la partie reconfigurable du SoC. Pour le SoC Xilinx Zynq-7000 il existe des registres de configuration liés à chacune des interfaces esclaves AXI GP et HP. Ces registres contrôlent les accès aux régions sécurisées de la mémoire externe depuis la partie reconfigurable du SoC. Par exemple, dans le cas de la figure 48, les registres de contrôle des interfaces esclaves AXI GP et HP connectées aux IP non-sécurisées (connexions rouges dans la figure 48) doivent être configurés pour interdire aux interfaces maîtres des IP non-sécurisées d'accéder à la région mémoire sécurisée (région mémoire GP1 en vert dans la figure 48) de la mémoire externe. Cette configuration réduit les menaces que représente des attaques par accès directes DMA depuis la partie reconfigurable que nous avons présenté dans le chapitre 3 de ce manuscrit.

Cette méthodologie en trois étapes permet aux concepteurs de renforcer la sécurité d'un SoC hétérogène complexe contre la majorité des attaques présentées dans ce manuscrit. Cependant, cela n'est pas suffisant, et des protections supplémentaires doivent être utilisées pour protéger le système contre les chevaux de Troie matériels insérés dans la partie reconfigurable du SoC. La suite de ce chapitre présente un système de vérification des règles de conception qui permet de vérifier les connexions entre les différents blocs du design.

2. Protection #2 : Vérification des règles de conception

Comme nous l'avons montré, la plupart des attaques présentées dans le deuxième chapitre de ce manuscrit peuvent être mise en place en exploitant un outil de CAO corrompu par un logiciel malveillant ou directement par un concepteur malveillant. Pour protéger la conception d'un SoC contre de telles menaces, l'outil de CAO et son utilisation nécessitent une protection logicielle dédiée. Le partage des fichiers entre développeurs doit être sécurisé à l'aide de fonctions cryptographiques (chiffrement, signature, etc.) et le concepteur doit obligatoirement vérifier les règles de conception. Cette vérification assure que le système conçu est conforme aux

spécifications fonctionnelles déclarées en amont de la conception. Elle peut aussi limiter l'insertion de chevaux de Troie matériel visant la AXI corruption des signaux du bus AXI.

La non-corruption des connexions entre les interfaces du système font partie des règles que doit vérifier le concepteur. Celui-ci doit s'assurer que tous les signaux des canaux AXI ne sont pas corrompus. Pour cela, nous proposons un programme python qui vérifie la connexion des signaux de sécurité entre les interfaces. Le code source est disponible en ligne [79]. Malgré le fait qu'il s'agit d'un code dédié à l'outil de CAO Vivado, ce code python peut facilement être adapté à d'autres outils de CAO.

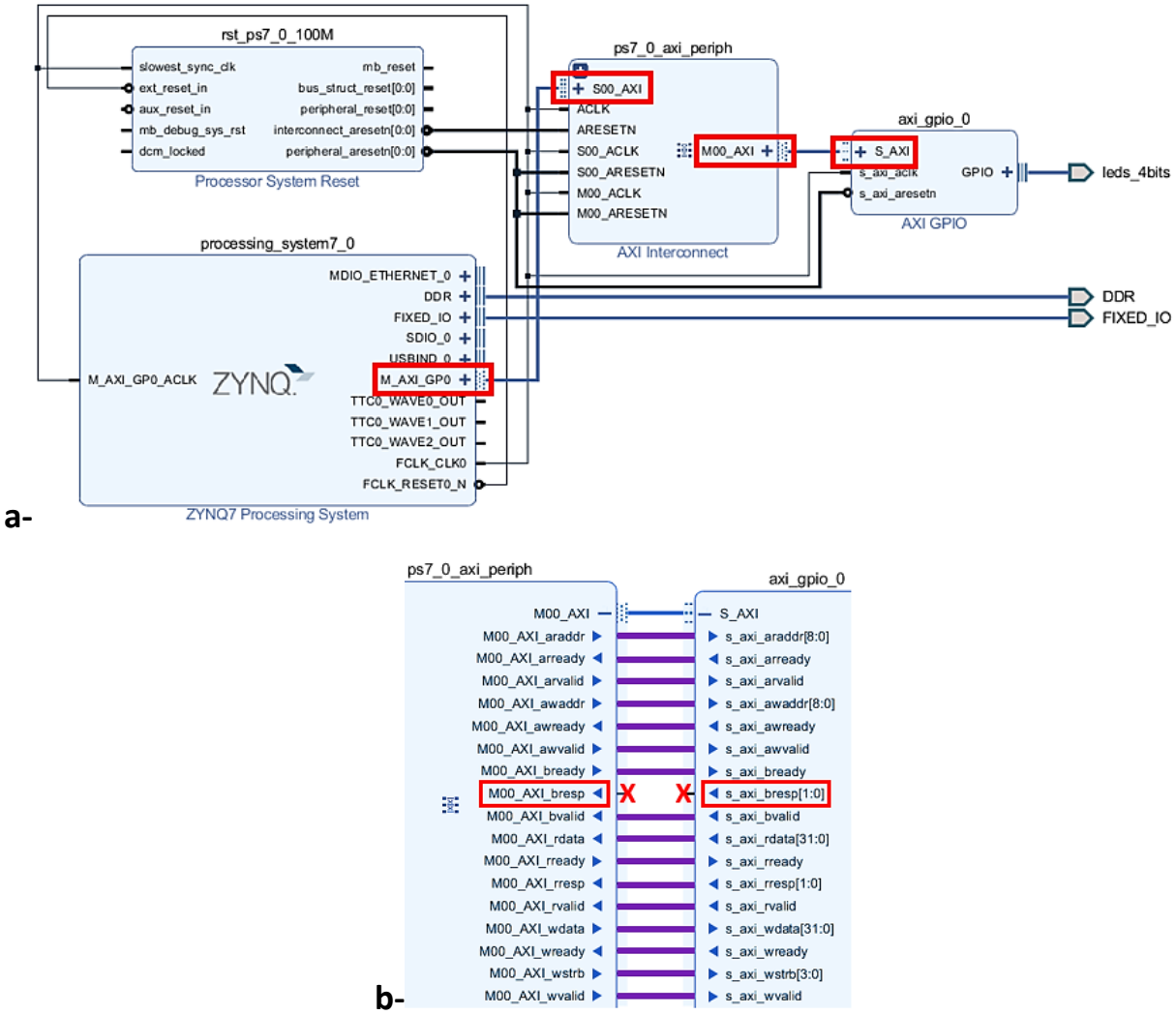

Figure 48: Vérification des règles de conception, a- design à vérifier , b- connexion corrompu entre l'interface M00_AXI de l'IP ps7_0_axi_periph et l'interfaces S_AXI de l'IP axi_gpio_0

Le programme python prend en entrée le code source VHDL du design et fournit en sortie la liste des connexions corrompues s'ils en existent. La liste contient les noms des signaux

corrompus avec le nom des deux interfaces affectées. Par exemple pour vérifier le design de la figure 49-a, Le programme python commence par recenser les différentes IP (entités) du design en utilisant le code source du top level puis il répertorie les interfaces AXI (rectangles rouge, figure 49-a), maîtres et esclaves, avec les signaux connectés à ses interfaces pour chaque IP. Ensuite, le programme prend deux par deux les interfaces connectées et vérifie les signaux qui les connectent. Dans notre exemple, la figure 49-b montre que la connexion entre le signal AXI_bresp (rectangle rouge, figure 49-b) de l'interface M00_AXI de l'IP ps7_0_axi_periph et celui de l'interfaces S_AXI de l'IP axi_gpio_0 est corrompu. Dans ce cas, le programme python renvoi le nom du signal corrompu et le nom des deux interfaces AXI affectées, ps7_0_axi_periph/M00_AXI et axi_gpio_0/S_AXI.

Cette solution doit être systématiquement ajoutée à tous les outils de CAO pour assurer l'intégralité du code source RTL et éviter toute modification malveillante du code source. Mais elle ne permet pas de régler tous les problèmes de sécurité et des protections complémentaires doivent être utilisées comme le chiffrement du bus AXI tel que la section suivant le présente.

3. Protection #3 : Chiffrement des transactions du bus AXI

Cette section présente une étude de faisabilité sur l'utilisation d'un système de chiffrement des données pour sécuriser les transactions sur le bus AXI et pour protéger ainsi l'échange de données entre deux entités dans un SoC hétérogène complexe. Dans cette section, l'algorithme de chiffrement de flux (appelé aussi chiffrement par flot) Trivium [80] est utilisé pour sécuriser les transactions entre une IP sécurisée de la partie reconfigurable du SoC et une ou des applications logicielles sécurisées exécutées dans la partie processeur du SoC. Les performances de l'implémentation logicielle et matérielle sur le SoC Xilinx Zynq-7000 seront étudiées.

Remarque : Cette section ne traite pas de la problématique de la gestion des clés cryptographiques et de l'échange de ces clés entre les parties processeur et reconfigurable du SoC.

3.1. Architecture logicielle et matérielle du système de chiffrement du bus AXI

Cette section utilise le design présenté dans la section chap3-1. Les différentes ressources du système hétérogène, notamment la partie processeur, la partie reconfigurable et la mémoire externe, sont partitionnées en ressources sécurisées et ressources non-sécurisées. Les IP

sécurisées de la partie reconfigurable sont implantées en utilisant la méthodologie en trois étapes présentée le chapitre 5 de ce manuscrit.

Pour chiffrer les transactions sur le bus AXI entre les parties processeur et reconfigurable du SoC hétérogène complexe, une application sécurisée intégrant un algorithme de chiffrement de flux est implémentée dans la partie processeur pour effectuer les opérations cryptographiques. Cette application est utilisée chaque fois qu'une transaction est destinée à une IP matérielle sécurisée. La clé cryptographique utilisée par l'application est enregistrée dans une région sécurisée de la mémoire externe. Dans la partie reconfigurable, une crypto-IP matérielle réalisant le même algorithme de chiffrement par flux est implantée dans la partie reconfigurable du SoC entre l'AXI Interconnect et l'IP sécurisée. Cette crypto-IP matérielle est composée de quatre blocs comme cela est présenté sur la figure 50. La crypto-IP matérielle dispose d'une interface maître pour communiquer avec l'IP sécurisée, d'une interface esclave pour échanger avec l'AXI Interconnect, d'un contrôleur pour le traitement des données et la planification des opérations cryptographiques, et d'un algorithme léger de chiffrement de flux pour chiffrer et déchiffrer les transactions.

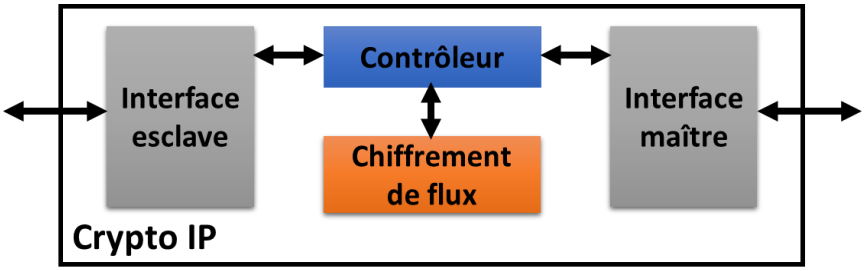

Figure 49: Architecture de la crypto IP matérielle implantée dans la partie reconfigurable du SoC

3.2. Le chiffrement léger de flux (Trivium)

Un chiffrement de flux, ou chiffrement en continue, est un algorithme de chiffrement à clé symétrique qui prend en entrée une clé et un vecteur d'initialisation, et fournit en sortie une chaîne de bits pseudo aléatoires (flux). Cette chaîne est ajoutée à un texte clair en utilisant l'opération logique ou-exclusif (xor) pour donner le texte chiffré. L'algorithme de chiffrement est généralement composé de registres à décalage et de fonctions de rétroaction linéaires et non-linéaires. Le processus de déchiffrement est analogue au chiffrement, il utilise la même clé et le même vecteur d'initialisation pour générer un flux pseudo-aléatoire identique à celui utilisé pour le chiffrement afin de récupérer le message original.

Trivium [80] est un algorithme de chiffrement de flux synchrone qui utilise une clé de 80 bits et un vecteur d'initialisation de 80 bits aussi. Son état interne est de 288 bits divisée sur trois registres à décalage de tailles différentes. Pour l'initialisation de Trivium, les trois registres à décalage sont initialisés à l'aide de la clé, le vecteur d'initialisation et des vecteurs de motif fixe. Ensuite, l'état interne est mis à jour 1152 fois (après 1152 périodes de l'horloge) de sorte que chacun de ses bits dépend de chaque bit de la clé et du vecteur d'initialisation d'une manière complexe non-linéaire.

Pour le chiffrement du bus AXI, nous utilisons Trivium car il est conçu pour fournir un compromis souple entre la vitesse et le nombre de portes logiques nécessaires à sa réalisation matérielle [80]. Un deuxième avantage est que la taille de sortie de Trivium peut être adapter à la taille du bus AXI (32 bits et 64 bits) utilisée dans le SoC Xilinx Zynq-7000, Trivium propose une taille de sortie allant de 1 à 64 bits. Dans ce chapitre, nous étudions aussi les implémentations de Trivium avec une taille de sortie de 8 bits et 16 bits pour présenter les performances de l'utilisation du chiffrement du bus de communication dans le cadre des microcontrôleurs de 8 bits et de 16 bits.

La suite de ce chapitre présente les performances des implémentations logicielles et matérielles de Trivium, une avec une taille de sortie de 8 bits, une avec 16 bits, une avec 32 bits et une avec 64bits.

3.3. Performances des implémentations logicielles et matérielles de Trivium

Cette section présente les performances des implémentations logicielles et matérielles de Trivium utilisant le SoC Xilinx Zynq-7000. Pour les expérimentations réalisées, la partie processeur du SoC Xilinx Zynq-7000 fonctionne à une fréquence de 650Mhz et la partie reconfigurable à une fréquence de 100 Mhz.

3.3.1. Performances des implémentations logicielles de Trivium

Dans cette partie, le code source de Trivium utilisé est basé sur l'implémentation logicielle de référence soumises au projet eSTREAM [81]. Elle est modifiée uniquement pour son adaptation au SoC Xilinx Zynq-7000 et aussi pour offrir différentes tailles de sortie : 8 bits, 16 bits et 32 bits, 64 bits. Les différentes implémentations logicielles de l'algorithme Trivium occupent un espace de 12 kilo-octets de la mémoire externe du SoC. La figure 51 donne le temps nécessaire

pour chiffrer 1 Méga-octet de données pour chacune des quatre implémentations logicielles (8 bits, 16 bits, 32 bits et 64 bits).

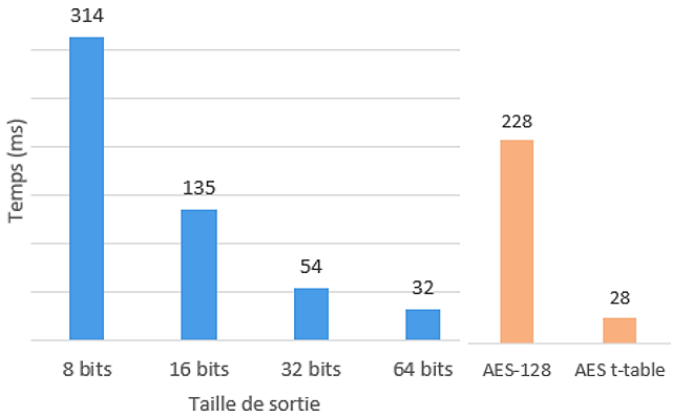

Figure 50: Temps nécessaire pour chiffrer 1 Méga-octets de données avec les quatre implémentations de Trivium (avec la taille de la sortie sur 8 bits, 16 bits, 32 bits et 64 bits), l'AES-128 et l'AES t-table exécutées sur la partie processeur du SoC cadencée à 650 MHz.

La figure 51 montre l'effet de la taille de sortie de Trivium sur le temps de génération du flux. Si nous utilisons l'implémentation 8 bits comme référence, la génération du flux est accéléré d'un facteur de 2.3 pour l'implémentation 16 bits, d'un facteur de 5.4 pour l'implémentation 32 bits et d'un facteur de 9.8 pour l'implémentation 64 bits. La figure 51 montre aussi que les implémentations, 16bits, 32bits, et 64 bits de Trivium sont rapides par rapport une implémentation logicielle de l'AES-128 mais lentes par rapport à une implémentation logicielle de l'AES t-table. La partie suivante présente les performances de l'implémentation matérielle de Trivium et les performances de l'implémentation matérielle de l'AES-128.

3.3.2. Performances des implémentations matérielles de Trivium

Dans cette section, les implémentations matérielles de l'algorithme de chiffrement de flux Trivium sont basées sur les implémentations publiées dans [80]. L'algorithme de chiffrement a été implémenté avec différente tailles de sortie (8 bits, 16 bits, et 32 bits, 64 bits). Les quatre implémentations matérielles de l'algorithmes Trivium sont au cœur de la Crypto-IP matérielle présentée à la figure 49. Le surcoût matériel ajouté par les deux interfaces maître et esclave, de la crypto-IP est de 105 LUT. Ce surcoût est fixe car il ne dépend pas de la taille de la sortie de l'algorithme Trivium. Le surcoût matériel ajouté par le contrôleur et par l'algorithme de chiffrement dépend lui de la taille de la sortie du chiffreur de flux.

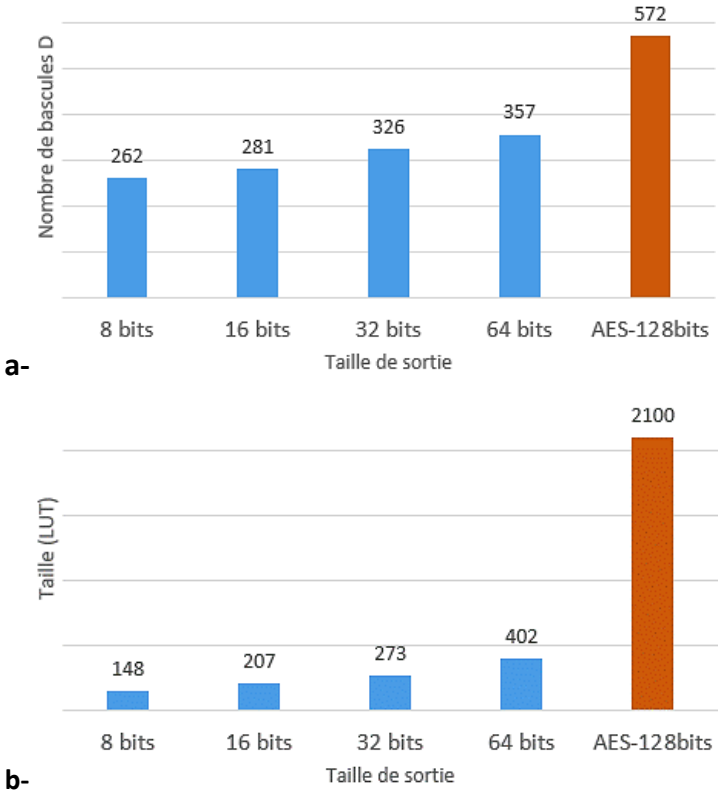

Figure 51: Performance des quatre implémentations matérielles de l'algorithme de chiffrement de flux Trivium (avec la taille de la sortie sur 8 bits, 16 bits, 32 bits et 64 bits) dans la partie reconfigurable du SoC, a- nombre de bascules D nécessaires à l'implémentation matérielle, b- nombre de LUT nécessaires à l'implémentation matérielle

Les figures 52-a et 52-b montrent que le nombre de bascules D et le nombre de LUT occupées par chacune des quatre implémentations matérielles de Trivium est systématiquement inférieure à celui nécessaire à une implémentation AES-128bits. Nous ajoutons aussi que chacune des quatre implémentations de Trivium n'utilisent qu'un très faible nombre des ressources disponibles dans la partie reconfigurable du SoC Xilinx Zynq-7000 qui contient 17 600 LUT et 35 200 bascules D.

3.4. Discussion

Les résultats précédents montrent que le surcoût global apporté par l'utilisation d'un chiffrement de flux pour chiffrer les transactions sur le bus AXI n'est pas très élevé en comparaison de la taille et des performances du SoC Xilinx Zynq-7000. Pour cette étude de faisabilité nous avons choisi d'utiliser l'algorithme de chiffrement par flux Trivium pour le chiffrement du bus AXI, mais le concepteur peut utiliser d'autres algorithmes de chiffrement de flux, comme par exemple Grain-128 [82]. Le concepteur peut aussi limiter l'utilisation du

chiffrement aux applications du monde sécurisé car ces applications n'effectuent pas un grand nombre de transactions et le système exécute les applications dans le monde non-sécurisé la majorité du temps. Le chiffrement des transactions du bus AXI est donc une solution efficace pour sécuriser les échanges entre les applications critiques du système et protéger les données privées.

4. Conclusion

Ce chapitre a présenté une méthodologie d'isolation des IP sécurisées dans un SoC FPGA embarquant la technologie ARM TrustZone. Le chapitre présente aussi un système de vérification des règles de conception notamment au niveau des connexions du bus AXI. Avec l'étude de chiffrement de flux et de leurs implémentation logicielles et matérielles, ce chapitre a montré la faisabilité du chiffrement du bus AXI avec un surcoût limité pour l'implémentation qu'elle soit logicielle ou bien matérielle.

Si nous faisons une synthèse en termes de sécurité apporté par les protections présentées dans ce chapitre nous pouvons dire que :

- Les protections #1 et #2, présentées dans ce chapitre, protègent contre les différentes attaques visant les signaux du bus AXI (attaques présentées dans le chapitre 2).
- Pour assurer l'efficacité des protections #1 et #2, il faut que les IP sécurisées soient de confiance comme cela est indiqué dans la section chap5-1. Effectivement, ces deux protections ne protègent pas contre les attaques basées sur des IP corrompus ou bien malveillants qui incluent des chevaux de Troie.
- La protection #3 protège contre l'attaque par FIFO présentée dans la section chap2-3.3.3, en général cette protection protège contre les attaques du type man-in-the-middle.
- Les protections proposées dans ce chapitre ne protègent pas contre les attaques DMA, les attaques par manipulation malicieuse du système DVFS et les attaques par exploitation malicieuse de la cohérence de cache. Proposer des protections contre ces attaques fait partie des perspectives de cette thèse.

Conclusion et perspectives

Ce manuscrit a présenté une évaluation en profondeur encore jamais réalisée de la sécurité de la technologie ARM TrustZone dans un SoC complexe hétérogène. Il a présenté de nombreuses attaques matérielles qui touchent différent éléments d'un SoC, comme le bus de communication AXI, la hiérarchie mémoire, le système de gestion d'énergie et le système de gestion de la cohérence de la mémoire cache. Différents scénarios d'attaques ont été présenté et certains implique la corruption du SoC lors de sa conception. Les attaques présentées exploitant un cheval de Troie matériel, une IP tiers malicieuse, un script malicieux ou un outil de CAO compromis par un logiciel malicieux ont une très grande portée. Elles doivent être absolument prises en compte par les concepteurs car elles peuvent conduire à une élévation de privilège, un déni de service, ou encore au vol de clés cryptographiques ou de données confidentielles. Les travaux réalisés dans le cadre de cette thèse de doctorat et présentés dans ce manuscrit prouvent que l'extension de la technologie TrustZone de ARM dans un SoC hétérogène complexe n'est pas sécurisée à l'heure actuelle. De plus, l'architecture des SoC hétérogène complexe disponible dans le commerce ne prend pas en compte les problématiques de sécurité et ouvre de nombreuses vulnérabilités alors même que le concepteur ayant mise en œuvre la technologie TrustZone de ARM au sein du SoC se pense protégé par celle-ci. C'est pourquoi, et les travaux développés durant cette thèse le montrent clairement, le déploiement de la technologie TrustZone de ARM dans des SoC hétérogènes complexes doit inévitablement s'accompagner de protections matérielles dédiées et de méthodologies de conception prenant en compte la sécurité du système et des données.

Pour répondre aux problématiques de sécurité du système et des données, nous avons présenté dans ce manuscrit des protections efficaces contre la majorité des attaques que nous avons étudié durant cette thèse. Nous avons proposé une méthodologie d'isolation matérielle renforcée pour sécurité la partie reconfigurable des SoC complexes et hétérogènes embarquant la technologie ARM TrustZone. Nous avons aussi apporté une preuve de faisabilité du chiffrement par flots du bus AXI.

Perspectives :

Les travaux présentées dans ce manuscrit ont introduit plus d'attaques que de contremesures. Pour cette raison, la continuité de ces travaux pourrait s'inscrire principalement dans la recherche de nouvelles contremesures qui puissent permettre de limiter les attaques DMA, les attaques par manipulation malicieuse du système DVFS et les attaques par exploitation malicieuse de la cohérence de cache. Nous pensons qu'il serait préférable de travailler sur le développement d'une nouvelle version de la technologie TrustZone dédiée aux SoC complexes hétérogènes qui intègre des protections matérielles et qui soit efficace à la fois contre les attaques matérielles et à la fois contre les attaques logicielles.

Nos travaux ont également mis en exergue l'augmentation de la complexité des chemins d'attaques (qui mixent l'exploitation de vulnérabilités logicielles et matérielles) conjointement à l'augmentation de la complexité des SoC. Il convient donc de continuer l'étude des chemins complexes d'attaque tout en proposant des solutions de sécurité qui protègent ces système complexes. L'utilisation d'une modélisation plus formelle des chemins d'attaque, par exemple en exploitant des arbres d'attaque et de défense, pourrait être une piste intéressante à explorer. Conjointement, la modélisation et la simulation complète d'une architecture complexe, par exemple avec un outil du type gem5, est aussi une piste prometteuse.

Un dernier aspect de notre travail est celui ayant trait aux vulnérabilités apportées par les outils de CAO pour les SoC hétérogènes et à leur sécurité. Le travail de thèse présenté dans ce manuscrit a démontré le danger que peut constituer un outil de CAO corrompu par un logiciel malveillant. Il convient de prendre en compte cette possibilité, d'étudier la possibilité de réaliser des logiciels malveillants ciblant les outils de CAO et de proposer des solutions efficaces pour limiter les vulnérabilités de ces derniers.

Publications et communications

Journal international avec comité de lecture

- El Mehdi Benhani, Lilian Bossuet, Alain Aubert. **''The Security of ARM TrustZone in a FPGA-Based SoC''**. IEEE Transactions on Computers 68(8): 1238-1248 (2019)

Conférence international avec comité de lecture

- El Mehdi Benhani, Cédric Marchand, Alain Aubert, Lilian Bossuet. **''On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC''**. SoCC 2017: 108-113
- El Mehdi Benhani, Lilian Bossuet. **''DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC''**. ICECS 2018: 489-492

Posters

- El Mehdi Benhani, Cuauhtemoc Mancillas-López, Lilian Bossuet. **'Secure Internal Communication of a Trustzone-Enabled Heterogeneous SoC Lightweight Encryption''**. FPT 2019: 239-242
- El Mehdi Benhani, Lilian Bossuet, Alain Aubert. **''L'évaluation de la sécurité de la technologie ARM TrustZone dans système sur puce hétérogène''**. Journée de la recherche de l'école doctorale EDSIS, Saint-Etienne, France, Juin 2019
- El Mehdi Benhani, Lilian Bossuet, Alain Aubert. **''L'évaluation de la sécurité de la technologie ARM TrustZone dans système sur puce hétérogène''**. Colloque national du GDR SoC2, Paris, France, juin 2018

Références bibliographies

[1] P. Kocher *et al.*, « Spectre attacks: Exploiting speculative execution », *arXiv preprint arXiv:1801.01203*, 2018.

[2] M. Lipp *et al.*, « Meltdown », *arXiv preprint arXiv:1801.01207*, 2018.

[3] S. Nie, L. Liu, et Y. Du, « Free-fall: hacking tesla from wireless to CAN bus », *Briefing, Black Hat USA*, p. 1–16, 2017.

[4] P. Kocher, J. Jaffe, B. Jun, et others, *Introduction to differential power analysis and related attacks*. 1998.

[5] S. Micali et L. Reyzin, « Physically observable cryptography », in *Theory of Cryptography Conference*, 2004, p. 278–296.

[6] N. Timmers, « Bypassing Secure Boot using Fault Injection », in *black hat*, 2016.

[7] B. Colombier, A. Menu, J.-M. Dutertre, P.-A. Moëllic, J.-B. Rigaud, et J.-L. Danger, « Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller. »

[8] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, et S. Ermeneux, « Laser-induced fault injection on smartphone bypassing the secure boot », *IEEE Transactions on Computers*, 2018.

[9] S. Tajik, H. Lohrke, J.-P. Seifert, et C. Boit, « On the power of optical contactless probing: Attacking bitstream encryption of FPGAs », in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, p. 1661–1674.

[10] A. Cui et R. Housley, « BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection », in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[11] N. Timmers, A. Spruyt, et M. Witteman, « Controlling PC on ARM using fault injection », in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2016, p. 25–35.

[12] R. Muresan et C. Gebotys, « Current flattening in software and hardware for security applications », in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2004, p. 218–223.

[13] K. Tiri, M. Akmal, et I. Verbauwhede, « A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards », in *Proceedings of the 28th European solid-state circuits conference*, 2002, p. 403–406.

[14] J. Irwin, D. Page, et N. P. Smart, « Instruction stream mutation for non-deterministic processors », in *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, 2002, p. 286–295.

[15] J. A. Ambrose, R. G. Ragel, et S. Parameswaran, « RIJID: random code injection to mask power analysis based side channel attacks », in *Proceedings of the 44th annual Design Automation Conference*, 2007, p. 489–492.

[16] S. Skorobogatov, « Physical attacks and tamper resistance », in *Introduction to Hardware Security and Trust*, Springer, 2012, p. 143–173.

[17] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre, et G. Gogniat, « ARMHEx: A hardware extension for DIFT on ARM-based SoCs », in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, p. 1–7.

[18] GlobalPlatform, « TEE Protection Profile, GlobalPlatform Device Committee TEE Protection Profile Version 1.2.1 », *https://www.globalpla-tform.org/specificationsdevice.asp*.
[19] GlobalPlatform, « TEE client API specification version 1.0 », *http://globalpla-tform.org*, 2010.
[20] A. ARM, « Security technology building a secure system using trustzone technology (white paper) », *ARM Limited*, 2009.
[21] L. Bossuet, « Sécurité des systèmes embarqués », 2018.
[22] U. Kanonov et A. Wool, « Secure containers in Android: the Samsung KNOX case study », in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2016, p. 3–12.
[23] QSEE, « QSEE, Retrieved from https://www.qualcomm.com/snapdragon/security », PhD Thesis.
[24] TRUSTONIC, « Security Target Kinibi v311A Security Target », PhD Thesis, 2017.
[25] SierraTEE, « Retrieved from http://www.openvirtualization.org/ », PhD Thesis, 2012.
[26] OP-TEE, « Project OP-TEE, GitHub repository », *https://github.com/OP-TEE*, 2017.
[27] B. McGillion T. Dettenborn, T. Nyman et N. Asokan, « Open-TEE - an open virtual trusted execution environment », PhD Thesis, 2015.
[28] TOPPERS, « SafeG, Retrieved from https://www.toppers.jp/en/safeg.html », PhD Thesis.
[29] S. Pinto et N. Santos, « Demystifying Arm TrustZone: A Comprehensive Survey », *ACM Computing Surveys (CSUR)*, vol. 51, n° 6, p. 130, 2019.
[30] S. Chow, P. Eisen, H. Johnson, et P. C. Van Oorschot, « White-box cryptography and an AES implementation », in *International Workshop on Selected Areas in Cryptography*, 2002, p. 250–270.
[31] Xilinx, « Zynq UltraScale+ MPSoC Technical Reference Manual UG1085 (v1.0) November 24, 2015 », 2015.
[32] Intel, « Intel Stratix 10 Hard ProcessorSystem Technical Reference Manual », 2019.
[33] V. Kindratenko et D. Pointer, « A case study in porting a production scientific supercomputing application to a reconfigurable computer », in *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, p. 13–22.
[34] A. Holdings, « AMBA AXI and ACE Protocol Specification », Tech. rep. 2011. url: http://infocenter. arm. com/help/topic/com. arm. doc …, 2013.
[35] A. Holdings, *ARM system memory management unit architecture specification—SMMU architecture version 2.0*. 2013.
[36] I. AMD et O. Virtualization, « Technology (IOMMU) Specification », 2007.
[37] Xilinx, « Zynq-7000 All Programmable SoC Technical Reference Manual, UG585 v1.11 », 2016.
[38] A. ARM, « Cortex-A9 TrustZone example », *http://infocenter.arm.co-m/help/topic/com.arm.doc.faqs/ka15417.html*, 2013.
[39] A. Moradi, A. Barenghi, T. Kasper, et C. Paar, « On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs », in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, p. 111–124.
[40] D. Guillaume et A. Iván, « Vulnerabilities in High Assurance Boot of NXP i.MX microprocessors », *https://blog.quarkslab.com/vulnerabilities-in-high-assurance-boot-of-nxp-imx-microprocessors.html*, 2017.
[41] O. Savry, T. Hiscock, et M. El Majihi, *Sécurité matérielle des systèmes: Vulnérabilité des processeurs et techniques d'exploitation*. Dunod, 2019.
[42] F. Majeric, B. Gonzalvo, et L. Bossuet, « Jtag combined attack-another approach for fault injection », in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2016, p. 1–5.
[43] F. Majeric, « Etude d'attaques matérielles et combinées sur les "System-on-chip" », PhD Thesis, 2018.

[44] A. Moradi, D. Oswald, C. Paar, et P. Swierczynski, « Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering », in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013, p. 91–100.
[45] A. Moradi et T. Schneider, « Improved side-channel analysis attacks on xilinx bitstream encryption of 5, 6, and 7 series », in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2016, p. 71–87.
[46] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, et G. Sigl, « How to break secure boot on fpga socs through malicious hardware », in *International Conference on Cryptographic Hardware and Embedded Systems*, 2017, p. 425–442.
[47] W. Li, Y. Xia, L. Lu, H. Chen, et B. Zang, « TEEv: virtualizing trusted execution environments on mobile platforms », in *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2019, p. 2–16.
[48] P. Carru, « Attack TrustZone with Rowhammer », *https://firmwares-ecurity.com/tag/rowhammer/*, 2017.
[49] V. Van Der Veen *et al.*, « Drammer: Deterministic rowhammer attacks on mobile platforms », in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, p. 1675–1689.
[50] J. A. Halderman *et al.*, « Lest we remember: cold-boot attacks on encryption keys », *Communications of the ACM*, vol. 52, n° 5, p. 91–98, 2009.
[51] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, et S. Mangard, « DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks », in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, p. 565–581.
[52] S. Chaudhuri, « A security vulnerability analysis of SoCFPGA architectures », in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, p. 1–6.
[53] B. Gras, K. Razavi, E. Bosman, H. Bos, et C. Giuffrida, « ASLR on the Line: Practical Cache Attacks on the MMU. », in *NDSS*, 2017, vol. 17, p. 26.
[54] D. Gruss, C. Maurice, K. Wagner, et S. Mangard, « Flush+ Flush: a fast and stealthy cache attack », in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016, p. 279–299.
[55] Y. Yarom et K. Falkner, « FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack », in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, p. 719–732.
[56] B. Gras, K. Razavi, H. Bos, et C. Giuffrida, « Translation leak-aside buffer: Defeating cache side-channel protections with ${$TLB$}$ attacks », in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, p. 955–972.
[57] D. A. Osvik, A. Shamir, et E. Tromer, « Cache attacks and countermeasures: the case of AES », in *Cryptographers' track at the RSA conference*, 2006, p. 1–20.
[58] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, et S. Mangard, « Armageddon: Cache attacks on mobile devices », in *25th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 16)*, 2016, p. 549–564.
[59] A. Tang, S. Sethumadhavan, et S. Stolfo, « CLKscrew: Exposing the Perils of Security-Oblivious Energy Management, Usenix 2018 (Distinguished Paper Award) », 2018.
[60] J. Krautter, D. R. Gnad, et M. B. Tahoori, « FPGAhammer: remote voltage fault attacks on shared FPGAs, suitable for DFA on AES », *IACR Transactions on Cryptographic Hardware and Embedded Systems*, p. 44–68, 2018.
[61] M. Zhao et G. E. Suh, « FPGA-based remote power side-channel attacks », in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, p. 229–244.
[62] L. Bossuet, P. Bayon, et V. Fischer, « Electromagnetic transmission of intellectual property data to protect FPGA designs », in *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*, 2015, p. 150–169.

[63] E. Benhani et L. Bossuet, « Design a TrustZone-enalble SoC usign Xilinx VIVADO CAD tool », Technical Report, University of Lyon, 2017. https://perso. univ-stetienne …, 2017.
[64] C. Benhani El mehdi, Marchand, A. Aubert, et L. Bossuet, « On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC », in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, 2017, p. 108–113.
[65] E. Benhani, L. Bossuet, et A. Aubert, « The Security of ARM TrustZone in a FPGA-based SoC », *IEEE Transactions on Computers*, vol. 68, n° 8, p. 1238–1248, 2019.
[66] « (UG835) Vivado Design Suite Tcl Command Reference Guide », *Xilinx Inc*, 2013.
[67] N. Fern, I. San, C. K. Koç, et K.-T. T. Cheng, « Hardware trojans in incompletely specified on-chip bus systems », in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, 2016, p. 527–530.
[68] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, et S. Capkun, « Thermal covert channels on multi-core platforms », in *24th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 15)*, 2015, p. 865–880.
[69] M. Alagappan, J. Rajendran, M. Doroslovački, et G. Venkataramani, « DFS covert channels on multi-core platforms », in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017, p. 1–6.
[70] L. Bossuet, P. Bayon, et V. Fischer, « An Ultra-Lightweight Transmitter for Contactless Rapid Identification of Embedded IP in FPGA », *IEEE Embedded Systems Letters*, vol. 7, p. 1-1, 2015, doi: 10.1109/LES.2015.2454236.
[71] L. Bossuet et others, « DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC », in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018, p. 489–492.
[72] M. Kim, S. Kong, B. Hong, L. Xu, W. Shi, et T. Suh, « Evaluating coherence-exploiting hardware trojan », in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, p. 157–162.
[73] J. Daemen et V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.
[74] D. J. Bernstein, « Cache-timing attacks on AES », 2005.
[75] B. Gülmezoğlu, M. S. Inci, G. Irazoqui, T. Eisenbarth, et B. Sunar, « A faster and more realistic flush+ reload attack on AES », in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2015, p. 111–126.
[76] E. Savaş et C. Yılmaz, « A generic method for the analysis of a class of cache attacks: a case study for AES », *The Computer Journal*, vol. 58, n° 10, p. 2716–2737, 2015.
[77] E. Tromer, D. A. Osvik, et A. Shamir, « Efficient cache attacks on AES, and countermeasures », *Journal of Cryptology*, vol. 23, n° 1, p. 37–71, 2010.
[78] E. Hallett, « Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools) ».
[79] L. B. Benhani El Mehdi, [En ligne]. Disponible sur: http://labh-curien.univ-st-etienne.fr/~bossuet/Security_design-rules_checker.zip.
[80] C. De Canniere et B. Preneel, « Trivium specifications », in *eSTREAM, ECRYPT Stream Cipher Project*, 2005.
[81] S. Babbage *et al.*, « The eSTREAM portfolio », *eSTREAM, ECRYPT Stream Cipher Project*, p. 1–6, 2008.
[82] M. Hell, T. Johansson, A. Maximov, et W. Meier, « A stream cipher proposal: Grain-128 », in *2006 IEEE International Symposium on Information Theory*, 2006, p. 1614–1618.