Exploitation malicieuse du mécanisme de cohérence de la mémoire cache dans un SoC complexe hétérogène

Ce chapitre présente l'exploitation malicieuse du mécanisme de cohérence de la mémoire cache d'un SoC complexe hétérogène depuis la partie reconfigurable. Le chapitre débute par une présentation des signaux de l'interface ACP qui contrôlent la cohérence d'une requête à la mémoire cache et les signaux de protection liés à la technologie ARM TrustZone. Ensuite, le chapitre présente une méthode pour mesurer le temps d'accès à une donnée cohérente depuis la partie reconfigurable du SoC et une méthode pour expulser une ligne de la mémoire cache depuis cette même partie reconfigurable. La maîtrise de ces deux méthodes est la condition sine qua non pour la mise en place d'une attaque par analyse temporelle de la mémoire cache.

La fin du chapitre présente trois scénario d'attaque exploitant malicieusement le mécanisme de cohérence de la mémoire cache depuis la partie reconfigurable d'un SoC complexe hétérogène : deux attaques par analyse temporelle de la mémoire cache ciblant l'exécution d'un algorithme de chiffrement symétrique AES exécuté dans le monde sécurisé par la partie processeur (attaques du type Flush+Reload et du type Evict+Time) et une attaque par transmission d'informations sensibles via un canal dérobé.

1. La cohérence de la mémoire cache depuis la partie reconfigurable dans un SoC complexe hétérogène

Dans un SoC complexe hétérogène, l'unité SCU maintient la cohérence de la mémoire cache entre les différents cœurs du processeur (comme cela est détaillé dans la section chap1-1.2.5). L'unité SCU arbitre les requêtes vers la mémoire cache L2 des cœurs du processeur qui sont issues des interfaces maîtres des accélérateurs matériels connectés à l'interface ACP (Accelerator Coherency Port). Une interface maître connectée à l'interface ACP peut lire la mémoire cohérente directement à partir des mémoires caches L1 et L2, mais ne peut pas écrire directement dans le cache L1. Les scénarios possibles de lecture et d'écriture d'une interface maître ACP sont les suivants :

- Requête de lecture avec des données cohérentes dans la mémoire cache L1 : l'interface maître ACP obtient les données directement de la mémoire cache L1.
- Requête de lecture avec des données cohérentes dans la mémoire cache L2 : la demande est mise en file d'attente dans le SCU et la requête est transmise à la mémoire cache L2.
- Requête de lecture avec des données cohérentes qui n'existent pas dans les mémoires cache L1 et L2 : selon l'état précédant des données, les mémoires cache L1 ou L2 peuvent lancer une requête pour obtenir les données demandées à la mémoire RAM ou bien à la mémoire principale avec la mise en attente de la requête initiale jusqu'à la disponibilité des données.
- Requête d'écriture avec des données cohérentes dans le cache L1: dans un premier temps les données au niveau de la mémoire cache L1 sont marquées comme non valides et les lignes de cache sont expulsées vers la mémoire cache L2. Puis, dans un second temps, la requête d'écriture est planifiée dans le SCU et les données sont écrites dans la mémoire cache L2. Enfin, lorsque le processeur accède à la même adresse mémoire que celle visée par la requête, un échec se produit dans la mémoire cache L1.
- Requête d'écriture avec des données cohérentes dans la mémoire cache L2 : la requête est planifiée dans le SCU puis les données sont écrites dans la mémoire cache L2.

- Requête d'écriture avec des données cohérentes qui ne sont pas dans les mémoires cache L1 et L2: la requête est destinée directement au contrôleur de la mémoire externe.

Ces scénarios sont contrôlés par les signaux AxCache[3:0] et AxUser[4:0] de l'interface ACP qui sont détaillés dans la section suivante.

1.1. Les signaux AxCache[3:0] et AxUser[4:0]

Les signaux AxCache[3:0] et AxUser[4:0] (pour chacun des signaux x = R pour lecture ou x = W pour écriture) de l'interface ACP permettent de contrôler la cohérence d'une requête vers les mémoires cache depuis la partie reconfigurable. Le signal AxUser[4:0] est composé du bit AxUser[0] qui est un bit de partage et des bits AxUser[4:1] qui contrôlent la stratégie d'écriture (Write-back [37], Write-through, … etc) adoptée par la requête. Les bits AxUser[4:0] ne sont pas interprétés par l'unité SCU et ils sont transmis directement au contrôleur de la mémoire cache L2 qui les utilise en cas d'une mémoire cache configurée en mode exclusif. Le signal AxCache[3:0] contrôle aussi la stratégie d'écriture adoptée par la requête. Il est composé des bits suivants :

- Le bit AxCache[0] qui contrôle la mise en mémoire tampon (Bufferable, B),
- Le bit AxCache[1] qui contrôle la mise en mémoire cache (Cacheable, C),
- Le bit AxCache[2] qui contrôle l'allocation sur lecture (Read-Allocate, RA),
- Le bit AxCache[3] qui contrôle l'allocation sur écriture (Write-Allocate, WA).

Selon les recommandations de ARM [68], le bit AxUser[0] et le bit AxCache[1] doivent être valides pour une requête ACP cohérente.

1.2. Le signal AxPort[2:0]

Le signal AxPROT[2:0] (pour lequel x = R pour lecture et x = W pour écriture) est un signal d'autorisation d'accès qui permet de protéger les interfaces esclaves des IP sécurisées contre les requêtes malveillantes. Il est composé des bits suivants :

- Le bit AxPROT[0] qui contrôle le niveau du privilège d'une requête,
- Le bit AxPROT[1] qui contrôle l'état de sécurité de la requête ACP,
- Le bit AxPROT[2] qui contrôle l'accès aux données ou aux instructions.

Les interfaces maîtres connectées à l'interface ACP contrôlent le signal AxPROT[2:0], en particulier le bit AxPROT[1] qui transmet l'état de sécurité de la requête à l'interface esclave. Dans un système intégrant la technologie ARM TrustZone, chaque interface maître impose la valeur du bit AxPROT[1], cela peut mettre en danger le système si l'interface maître est exploitée malicieusement par un cheval de Troie. De plus, ce scénario est particulièrement crédible lorsque les interfaces esclave de la partie processeur ne sont pas configurée pour interdire l'accès aux régions sécurisées de la mémoire principale depuis la partie reconfigurable ce qui est souvent le cas. La suite de ce chapitre présente différents types d'attaque exploitant la cohérence de la mémoire cache.

2. Les éléments de l'attaque

La section suivante présente les éléments utilisés pour la mise en place des trois attaques présentées à la fin de ce chapitre. Cette section présente tout d'abord les méthodes utilisées pour distinguer l'échec d'un succès d'une requête à la mémoire cache depuis la partie reconfigurable ou depuis la partie processeur. La section présente également les méthodes utilisées pour expulser une ligne de la mémoire cache ciblée par l'attaque. La maîtrise de ces méthodes est indispensable à la réalisation des attaques.

2.1. Distinguer un échec d'un succès de la mémoire cache

La première condition pour réussir une attaque visant la mémoire cache consiste à distinguer l'échec d'un succès d'une requête adressée à cette mémoire cache. Cette section présente des méthodes à utiliser pour distinguer l'échec d'un succès depuis la partie reconfigurable ou bien depuis la partie processeur.

2.1.1. Depuis la partie reconfigurable

La méthode présentée dans cette partie se base sur la mesure du temps d'accès à une donnée depuis la partie reconfigurable pour distinguer l'échec d'un succès d'une requête à la mémoire cache. Cette méthode utilise les signaux du bus AXI (détaillés dans la section chap1-1.1) connectant l'interface maître à l'interface ACP pour mesurer le temps d'accès. Mais avant d'utiliser cette méthode, il faut s'assurer que l'interface maître est connectée à l'interface ACP comme cela est indiqué dans la section chap3-1, que les bit AxUser[0] et AxCache[1] soient valides, et que l'unité SCU et le *symmetric multiprocessor system* (SMP) soient activés. La suite de cette section présente les canaux du bus AXI qui fuitent des informations sur la présence ou pas d'une donnée dans la mémoire cache et la méthode utilisée pour mesurer le temps d'accès.

Durant une requête d'écriture cohérente depuis la partie reconfigurable, l'interface maître envoie l'adresse dans le canal d'adresse d'écriture du bus AXI suivie de la donnée à écrire dans le canal d'écriture de données. Une fois la donnée reçue par l'interface ACP, cette dernière envoie une réponse à la requête à l'interface maître en utilisant le canal d'écriture de réponse. En mesurant le nombre de cycles d'horloge écoulés entre le handshake du canal d'adresse d'écriture et le canal d'écriture de réponse durant une requête d'écriture, on constate que le nombre de cycles d'horloge ne varie pas par rapport à la présence de la donnée dans la mémoire cache L1 ou L2. Donc, il n'est pas possible de distinguer un échec d'un succès lors d'une requête d'écriture.

Pour une requête de lecture cohérente depuis la partie reconfigurable, l'interface maître envoie tout d'abord l'adresse à lire dans le canal d'adresse de lecture du bus AXI. Ensuite, si la communication est établie avec l'interface ACP, cette dernière envoie la donnée à l'interface maître en utilisant le canal de lecture de donnée. En mesurant le nombre de cycles d'horloge écoulés entre le handshake du canal d'adresse de lecture et celui du canal de lecture de donnée, le nombre de cycles d'horloge varie selon l'existence ou pas de la donnée dans la mémoire cache L1 ou L2. Il est court, si la donnée réside dans la mémoire cache et il est long dans le cas contraire. Pour distinguer les succès d'un échec d'une requête à la mémoire cache depuis la partie reconfigurable, il suffit donc de compter le nombre de cycles d'horloge entre les deux handshake des deux canaux du bus AXI utilisés dans une requête de lecture.

Dans la suite de ce chapitre, l'algorithme 4 est utilisé pour valider les méthodes de mesure de temps et d'expulsion de la ligne de cache introduites dans ce chapitre. L'algorithme effectue en boucle, un accès à l'adresse donnée en entrée, une expulsion de la ligne de cache contenant cette adresse une itération sur deux, et une mesure de temps d'accès à cette adresse. L'algorithme 4 provoque un échec de la mémoire cache toutes les deux itérations en expulsant la ligne de cache.

Algorithme 4: Vérification de la méthode utilisée pour distinguer un échec d'un succès

```
Input: adresse
    for i = nombre_dacces to 0 do
        Accès(adresse)
        if !(i%2) then
          Expulser_ligne_cache (adresse)
        end if
        Measure_clock_cycles (Access(adresse))
    end for
```

La méthode utilisée dans la fonction ***Expulser_ligne_cache*** est détaillée dans la section chap4-2.2.1 et la méthode utilisée dans la fonction ***Measure_clock_cycles*** est celle basée sur la mesure de nombre de cycles d'horloge entre le handshake du canal d'adresse de lecture et le canal de lecture de donnée.

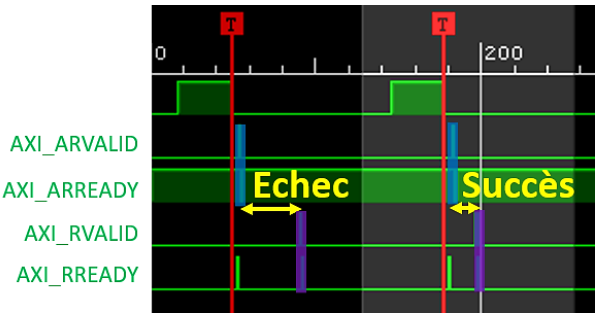

Figure 38: Un échec et un succès d'une requête de lecture cohérente

La figure 39 présente la capture des signaux du handshake (les signaux _valid et _ready) du canal d'adresse de lecture et du canal de lecture de donnée durant deux itérations successives de l'algorithme 4. La figure montre que le temps écoulé entre le handshake du canal de lecture d'adresse (AXI_ARVALID == '1' && AXI_ARREADY == '0', ligne verticale bleue figure 39) et celui du canal de lecture de donnée (AXI_RVALID == '1' && AXI_RREADY == '1', la ligne verticale violette figure 39) est plus long pour un échec que pour un succès. La figure 40 valide la méthode introduite dans cette section, les échecs et les succès de la mémoire cache sont clairement distinguables depuis la partie reconfigurable.

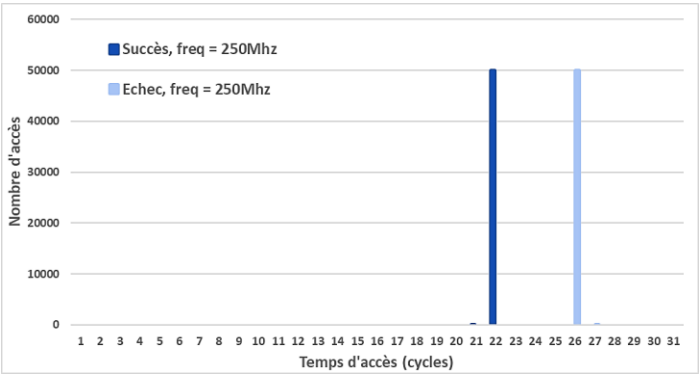

Figure 39: Histogramme des échecs et des succès d'une requête de lecture cohérente (Fréquence FPGA = 250Mhz)

Pour utiliser cette méthode dans d'autres systèmes, il faut réaliser une pré-étape de profilage (utiliser simplement l'algorithme 4) pour déterminer un seuil qui permet de différencier les échecs des succès. Comme nous l'avons vu dans le chapitre précédent, dans les SoCs modernes

il est possible de modifier la fréquence en cours d'exécution pour adapter dynamiquement la consommation de puissance du système. La fréquence d'horloge peut donc évoluer au cours du temps. Il faut donc réaliser le profilage à chaque changement de fréquence. La figure 41 montre que pour le SoC Xilinx Zynq-7000, chaque fréquence définie un seuil différent.

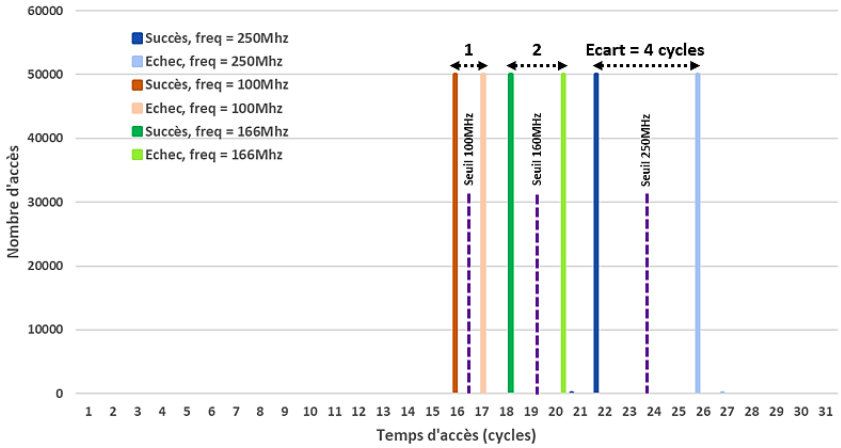

Figure 40: L'effet de la fréquence sur le seuil

La figure 41 montre que l'écart entre le nombre de cycles d'horloge pour les échecs et les succès diminue avec la fréquence d'horloge de la partie reconfigurable. Dans le SoC Xilinx Zynq-7000, l'écart passe de 4 cycles pour une fréquence = 250Mhz à un cycle pour une fréquence = 100Mhz. Cet écart est nul pour toutes les fréquences en dessous de 55Mhz, donc pour le SoC Xilinx Zynq-7000 cette méthode est limitée aux fréquences supérieures à 55Mhz. Pour l'utiliser dans un scénario d'attaque il faut prendre en considération cette limite.

La suite du chapitre démontre qu'il est possible de réaliser une attaque par analyse de la mémoire cache avec cette méthode de mesure du temps d'accès même avec un écart d'un cycle d'horloge (pour une fréquence d'horloge comprise entre 55 Mhz et 125 Mhz) entre les échecs et les succès d'une requête adressée à la mémoire cache par la partie reconfigurable.

2.1.2. Depuis la partie processeur

Pout distinguer l'échec d'un succès d'une requête à la mémoire cache depuis la partie processeur, l'utilisateur peut utiliser une des deux méthodes suivantes :

- L'utilisation du PMU (Performance Monitor Unit) qui offre 6 compteurs configurables pour compter le nombre des évènements, comme les échecs de la mémoire cache et le nombre d'accès à cette mémoire. Le PMU est accessible depuis l'espace utilisateur seulement si la configuration du système l'autorise, ce qui n'est pas le cas dans la

majorité des systèmes d'exploitation. Pour l'utiliser dans une attaque, il suffit d'initialiser les compteurs avant le lancement du programme victime et lire la valeur des compteurs une fois le programme victime terminé. Ensuite, il reste à analyser ces valeurs selon le type d'attaque mise en place.

- Mesurer le temps d'accès à une donnée dans le système mémoire. Pour mesurer le temps lorsque le système d'exploitation Linux est mise en œuvre dans le SoC, l'attaquant peut utiliser soit la fonction ***POSIX clock_gettime()***, soit l'appel système ***perf_event_open*** si le système autorise son utilisation vu qu'elle fait appelle au PMU, soit l'utilisation d'un thread qui incrémente une variable globale en boucle [58]. Cette méthode elle aussi demande une pré-étape de profilage pour chaque système pour déterminer un seuil qui permet de différencier les échecs des succès de la mémoire cache.

Comme dans la méthode précédente, pour chaque système et à chaque changement de fréquence, il faut réaliser une pré-étape de profilage (utilisation de l'algorithme 4) pour déterminer le seuil qui permet de distinguer l'échec d'un succès de la mémoire cache. Cependant, ce n'est pas que la fréquence et le changement de système qui affectent le choix du seuil. Ce seuil varie également en changeant la politique de remplacement d'une ligne de la mémoire cache. La figure 42 montre que dans le SoC Xilinx Zynq-7000, la politique de remplacement round-robin ajoute de la latence à l'accès aux données par rapport à une politique de remplacement aléatoire.

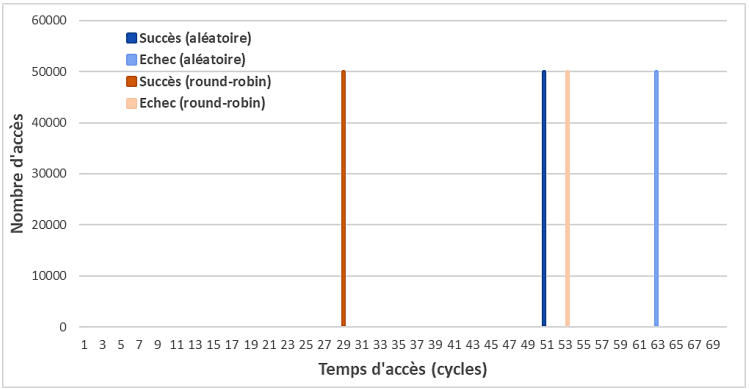

Figure 41: Histogramme des échecs et des succès avec deux politiques de remplacement différentes (aléatoire, round-robin)

2.2. Opération de maintenance de la mémoire cache

La deuxième condition pour réussir une attaque visant la mémoire cache consiste à utiliser une opération de maintenance privilégiée ou non-privilégiée pour expulser une ligne de la

mémoire cache ou pour invalider la totalité des lignes de cache. Cette section présente des méthodes à utiliser pour expulser une ligne de cache depuis la partie reconfigurable et dans la partie processeur.

2.2.1. Expulser une ligne de cache depuis la partie reconfigurable

Comme présenté dans les scénarios possibles de lecture et d'écriture d'une interface maître ACP dans la section chap4-1, une requête d'écriture cohérente force la mémoire cache L1 à invalider la ligne de cache contenant l'adresse de la requête, si la donnée cohérente réside dans la mémoire cache L1. Donc, il suffit d'envoyer une requête d'écriture cohérente pour expulser une adresse de la mémoire cache. Cependant, il faut ajouter à la requête une modification du signal WSTRB[4:0] de l'interface ACP qui contrôle le nombre d'octets valides dans le signal WDATA[31:0] et qui doivent être mis à jour dans la mémoire. Selon [72], il faut que les bits du signal soient à un état logique bas pour que les octets du signal WDATA[31:0] soient invalides. Avec cette modification, la requête d'écriture est valide, mais la valeur de l'adresse n'est pas mise à jour.

Cependant, cette technique n'arrive pas à expulser les lignes de cache si la région contenant l'adresse ciblée a une stratégie d'écriture autre que Write-back. Pour vérifier la possibilité d'utiliser cette technique d'expulsion dans une attaque de cache contre une région de mémoire, il faut utiliser l'algorithme 4 en remplaçant la fonction ***Expulser_ligne_cache*** par une requête d'écriture cohérente avec WSTRB[4:0] = b''0000''.

2.2.2. Expulser une ligne de cache depuis la partie processeur

Les processeurs ARM ont des instructions de maintenance de la mémoire cache accessibles depuis le coprocesseur CP15, mais ces instructions ne sont pas accessibles depuis l'espace utilisateur. Pour cette raison, l'attaquant peut utiliser l'une des stratégies d'expulsion présentes dans la littérature, comme celle présentée dans [58]. Dans les SoCs avec une architecture de cache simple, comme le SoC Xilinx Zynq-7000, il suffit d'accéder à un nombre N d'adresses congruentes à l'adresse à expulser. Dans un cache *n-way associative*, les adresses congruentes sont situées à une adresse à une distance proportionnelle à la **''taille du cache/nombre de *way*''** de l'adresse à expulser.

Le choix du nombre N dépend de la politique de remplacement de ligne de cache. Si la mémoire utilise une politique de remplacement round-robin, un nombre N égale au nombre de *way* est suffisant pour expulser une adresse car un cache *n-way associative* ne peut pas contenir

plus que N adresses congruent à la fois. Si la mémoire utilise une politique de remplacement aléatoire, le nombre N doit être supérieur au nombre de *way* pour parvenir à expulser l'adresse cible.

2.2.3. Invalider la totalité de la mémoire cache dans la partie processeur

Pour invalider simultanément la totalité des lignes de la mémoire cache dans la partie processeur, l'attaquant peut effectuer un changement de contexte (commutation entre deux processus du système d'exploitation). En effet, lors d'un changement de contexte, certains systèmes d'exploitation et mécanismes matériels vident automatiquement la mémoire cache pour éviter les fuites de données entre les processus.

3. Scénario d'attaque

Cette section présente trois scénarios d'attaques complètes (Flush+Reload, Evict+Time et une attaque par canal dérobé) exploitant la cohérence de cache. Les attaques sont implémentées en utilisant le SoC Xilinx Zynq-7000 qui inclut un processeur dual-cœur ARM Cortex-A9, une mémoire cache L1 4-*way set-associative* pour les instructions, et une pour les données avec une taille de 32Ko chacune. Le SoC possède également une mémoire cache L2 8-*way set-associative* de taille 512Ko et d'une ligne de cache de longueur de 32 octets. La mémoire cache L2 constitue le dernier niveau de mémoire cache LLC (Last Level Cache) pour le SoC Xilinx Zynq-7000, elle est partagée entre les deux cœurs ARM Cortex-A9 et les interfaces maîtres de la partie reconfigurable connectées à l'interface ACP.

Le design expérimental de la section chap3-1 est utilisé pour l'implémentation des trois attaques. La fréquence de la partie processeur est de 650Mhz et celle de la partie reconfigurable est de 250Mhz. Le signal AxProt[1] de l'interface ACP est fixé à un état logique bas pour permettre aux interfaces maîtres d'accéder aux régions sécurisées de la mémoire principale. Ils peuvent ainsi avoir accès aux données sensibles du système. La partie processeur autorise l'accès aux régions sécurisées de la mémoire principale depuis la partie reconfigurable.

Remarque : l'ensemble des attaques présentées dans cette section sont réalisées à l'aide d'une application bare-metal. Donc, les résultats présentés ne contiennent pas beaucoup de bruit contrairement à ce que l'on pourrait obtenir avec un système d'exploitation mis en œuvre dans le SoC.

3.1. Attaque par canal auxiliaire

Cette section présente deux attaques par analyse temporelle de la mémoire cache, l'attaques Flush+Reload et l'attaque Evict+Time, qui cible un algorithme de chiffrement symétrique. La section présente l'algorithme de chiffrement utilisé dans les attaques, ainsi que le modèle de menaces et les deux attaques.

3.1.1. AES-128 T-table

L'implémentation AES-128 T-table est une implémentation optimisée en termes de performances du standard AES-128 (Advanced Encryption Standard) [73]. Cette implémentation combine les trois fonctions d'un tour de l'AES (SubBytes, ShiftRows et MixColumns) en une seule étape en utilisant quatre tables de recherches précalculées T_0, T_1, T_2 et T_3 de 1ko (256 éléments de 32bits) pour les 9 premiers tours de l'algorithme. Le dernier tour utilise une table de recherches précalculées T_4 supplémentaire car ce tour n'utilise pas l'opération MixColumns. L'AES-128 T-table prend en entrée un texte clair de 16 octets $p = (p_0, p_1, \dots, p_{15})$, et une clé de 16 octets $k = (k_0, k_1, \dots, k_{15})$. L'équation (3) présente la structure du tour i ($1 \leq i \leq 9$) qui prend en entrée l'état intermédiaire $S_i = (s_{i,0}, s_{i,1}, \dots, s_{i,15})$ et la clé du tour $k_i = (k_{i,0}, k_{i,1}, \dots, k_{i,15})$. Le tour donne en sortie l'état intermédiaire S_{i+1}. L'état intermédiaire S_1 prend en entrée $p \oplus k = (p_0 \oplus k_0, p_1 \oplus k_1, \dots, p_{15} \oplus k_{15})$ et la clé secrète k.

$$
\begin{aligned}
S_{i+1} = \{ & T_0[s_{i,0}] \oplus T_1[s_{i,5}] \oplus T_2[s_{i,10}] \oplus T_3[s_{i,15}] \oplus \{k_{i,0}, k_{i,1}, k_{i,2}, k_{i,3}\}, \\
& T_0[s_{i,4}] \oplus T_1[s_{i,9}] \oplus T_2[s_{i,14}] \oplus T_3[s_{i,3}] \oplus \{k_{i,4}, k_{i,5}, k_{i,6}, k_{i,7}\}, \\
& T_0[s_{i,8}] \oplus T_1[s_{i,13}] \oplus T_2[s_{i,2}] \oplus T_3[s_{i,7}] \oplus \{k_{i,8}, k_{i,9}, k_{i,10}, k_{i,11}\}, \\
& T_0[s_{i,12}] \oplus T_1[s_{i,1}] \oplus T_2[s_{i,6}] \oplus T_3[s_{i,11}] \oplus \{k_{i,12}, k_{i,13}, k_{i,14}, k_{i,15}\}\}
\end{aligned}
\tag{3}
$$

L'AES-128 T-table est la cible de plusieurs attaques de cache [57], [74], [75]. Ces attaques exploitent le fait que l'état intermédiaire S_1 dépend directement du texte clair p et de la clé k. Si l'attaquant connait le texte claire p_i et les éléments (adresses) des tables de recherches (T_0, T_1, T_2 et T_3) utilisés durant l'opération, il peut déduire facilement l'octet k_i de la clé. Cette section utilise ce scénario d'attaque de cache pour démontrer la faisabilité d'une attaque par analyse de la mémoire cache depuis la partie reconfigurable du SoC.

3.1.2. Modèle de menaces

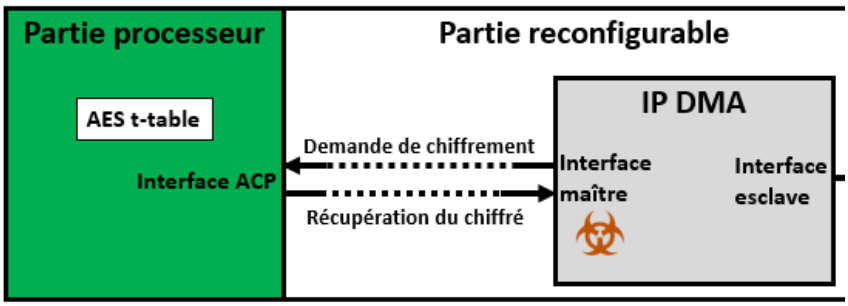

Figure 42: Schéma de l'attaque par canal auxiliaire

La figure 43 présente le modèle de menaces utilisé pour l'attaque Flush+Reload et l'attaque Evict+Time. Dans la partie reconfigurable du SoC Xilinx Zynq-7000, une IP DMA récupère des données à chiffrer d'un périphérique d'E/S et les transmet à la partie processeur pour les chiffrer à l'aide d'une implémentation de l'AES-128 T-table vulnérable aux attaques par analyse temporelle de la mémoire cache. Par la suite, l'IP DMA récupère les données chiffrées à la sortie de l'algorithme de chiffrement pour le transmettre au périphérique d'E/S. L'IP DMA n'a aucune information sur la clé secrète utilisée dans la partie processeur. L'interface maître de l'IP DMA inclus un cheval de Troie matériel qui exploite la cohérence de cache pour récupérer la clé de chiffrement puis la transmettre à l'extérieur du SoC. Le cheval de Troie a la possibilité d'initialiser des chiffrements depuis la partie reconfigurable en envoyant un texte clair.

Parmi les challenges rencontrés pour mettre en œuvre une attaque par l'analyse de la mémoire cache depuis la partie reconfigurable du SoC, il y a la recherche de l'emplacement des quatre tables de recherches dans la mémoire principale et l'alignement des tables $T_0, T_1, T_2\ et\ T_3$dans la mémoire. Dans les attaques présentées dans cette section les adresses des tables sont à fournir à l'interface maître au début de l'attaque. Mais si les adresses ne sont pas fournies, l'interface maître doit parcourir la totalité de la mémoire principale à la recherche des 4 premiers éléments de chaque table pour les localiser avant de lancer l'attaque.

3.1.3. Attaque Flush+Reload

L'attaque Flush+Reload présentée vise le premier tour de l'AES-128 T-table exécuté dans le monde sécurisé de la partie processeur. L'interface maître utilise l'algorithme 4 pour définir le seuil qui permet de distinguer les échecs des succès des requêtes à la mémoire cache avant chaque attaque.

Depuis la partie reconfigurable, l'interface maître déclenche l'exécution dans la partie processeur du chiffrement de 256 textes clairs aléatoires avec l'octet p_i fixe Avant chaque chiffrement, l'interface maître expulse la ligne de cache contenant un élément parmi les 256 éléments de la table T_0 si cet élément est dans la mémoire cache. Il mesure alors le temps d'accès à cet élément après la réception du texte chiffré. A l'aide du seuil et du temps d'accès aux éléments de la table T_0, l'interface maître peut en déduire quels sont les éléments de la table T_0 utilisés par le chiffrement et donc l'octet k_i de la clé [57].

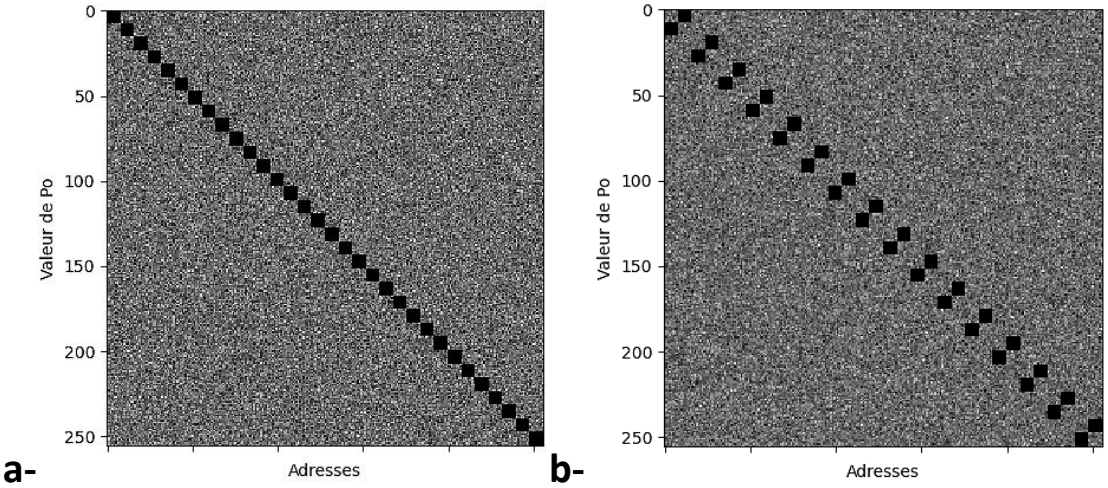

Figure 43: Attaque Flush+Reload exploitant la cohérence de cache,
a- fréquence = 250Mhz et $k_0 = 0x00$, b- fréquence = 100Mhz et $k_0 = 0x0f$

La figure 44-a présente la matrice de paterne d'accès aux adresses de la table T_0 pour $k_0 = 0x00$ et une fréquence de fonctionnement de 250Mhz dans la partie reconfigurable. Cette matrice est obtenue en ciblant les adresses de la table T_0 avec l'attaque Flush+Reload pour les 256 valeurs de p_i. La matrice de la figure 44-a présente une diagonale car l'état intermédiaire S_1 de l'algorithme de chiffrement accède à la valeur $T_0[p_i = 0x00 \oplus p_i]$. Dans le cas d'une taille de ligne de cache égale à 32 octets, la matrice révèle que les 5 bits supérieurs de l'octet k_0 [57] et cela réduit l'espace de recherche de la clé à 48 bits. Pour récupérer la totalité de la clé, il faut suivre l'une des méthodes présentées dans [71], [75]–[77].

La figure 44-b présente la matrice de paterne d'accès aux adresses de la table T_0 pour $k_0 = 0x0f$ avec une fréquence d'horloge de 100Mhz pour la partie reconfigurable du SoC. Cette matrice nous montre qu'il est possible de réaliser l'attaque de la mémoire cache à l'aide de la méthode présentée dans la section Chap4-2.1.1, ceci même avec une différence temporelle de seulement un cycle d'horloge entre un échec et un succès de la mémoire cache.

3.1.4. Evict+Time

L'attaque Evict+Time que nous avons réalisée vise aussi le premier tour de l'AES-128 T-table exécuté dans le monde sécurisé de la partie processeur. Les étapes principales de l'attaque Evict+Time réalisée sont :

Etape 1 : Depuis la partie reconfigurable, l'interface maître déclenche l'exécution dans la partie processeur du chiffrement d'un texte clair. Cette première opération charge dans la mémoire cache les éléments nécessaires pour la réalisée notamment les T-tables.

Etape 2 : Toujours depuis la partie reconfigurable, l'interface maître expulse la ligne de cache contenant un élément de la table T_0 si celui-ci réside dans la mémoire cache.

Etape 3 : L'interface maître déclenche à nouveau l'exécution dans la partie processeur du chiffrement du même texte clair. Maintenant, cette opération charge la mémoire cache avec seulement les éléments manquants pour la réaliser. La table T_0 qui a été retirée dans l'étape précédente sera donc chargée uniquement si elle est utilisée.

Etape 4 : L'interface maître mesure alors le temps écoulé entre le déclenchement du chiffrement et la réception du texte chiffré. Si le temps de chiffrement est supérieur à un certain seuil, l'interface maître peut en déduire que l'élément de la table T_0 a été utilisé puisque l'allongement du temps tient au chargement de la table dans la T_0 mémoire cache et ainsi trouver l'octet k_i de la clé.

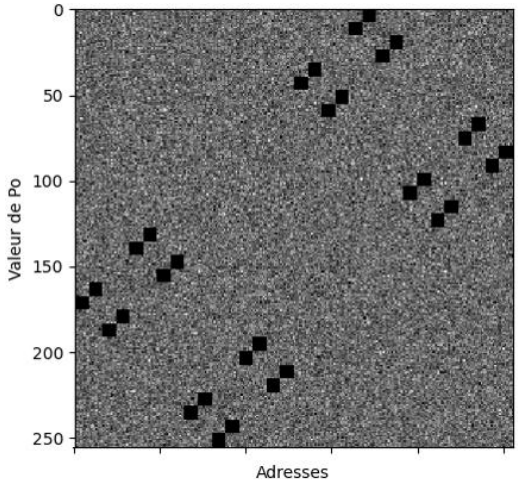

Figure 44: Attaque Evict+Time depuis la partie reconfigurable $k_0 = 0x51$

La figure 45 présente la matrice de paterne d'accès aux adresses de la table T_0 pour $k_0 = 0x51$ avec une fréquence de fonctionnement de 250Mhz pour la partie reconfigurable du SoC.

Cette matrice est obtenue en utilisant l'attaque Evict+Time. Elle valide la possibilité de cette attaque depuis la partie reconfigurable du SoC.

3.2. Attaque par canal dérobé

Dans cette attaque, le processus espion est une application logicielle malicieuse exécutée dans le monde sécurisé de la partie processeur du SoC. Le processus récepteur est un cheval de Troie matériel inséré dans la partie reconfigurable du SoC au niveau de l'interface maître de l'IP DMA. Le processus espion et le processus récepteur n'ont normalement pas le droit de communiquer directement. Cependant, ils utilisent tous les deux une adresse mémoire située dans une région sécurisée de la mémoire principale pour communiquer. Cette adresse est accessible uniquement en lecture par le processus récepteur et elle est accessible à la fois en lecture et en écriture par le processus espion.

Le processus espion transmet de l'information en chargeant ou en expulsant l'adresse partagée par les deux processus de la mémoire cache. Ainsi il peut coder l'information au travers de cette action. L'algorithme 5 présente l'encodage des données par le processus espion. Pour transmettre un 1 logique, le processus espion expulse l'adresse de la mémoire cache suivi d'une temporisation *Tempo_1*. Pour transmettre un 0 logique, le processus espion expulse l'adresse de la mémoire cache suivi d'une temporisation *Tempo_2*. La temporisation *Tempo_1* est plus lente que la *Tempo_2.* Entre deux bits successifs envoyés, le processus espion charge la donnée dans la mémoire cache suivi d'une temporisation *Tempo_3*. Le choix des temporisations *Tempo_1*, *Tempo_2 et Tempo_3* joue sur la bande passante et le taux d'erreur. La méthode d'encodage présentée dans cette section permet de réduire le taux d'erreur, même si la bande passante est très faible (1Mbs).

Le processus récepteur effectue une opération de lecture en continue à l'adresse partagée par les deux processus pour détecter les échecs et les succès de sa requête à la mémoire cache et les décoder pour récupérer l'information transmise par le processus espion. Le processus récepteur utilise l'algorithme 6 pour décoder les informations dérobées. A la détection d'un échec, le processus récepteur enregistre la valeur 1 dans le tableau de données. S'il détecte un succès, il enregistre la valeur 0. La figure 46 présente un exemple de décodage du message ''hi'', les symboles « larges » dans la figure sont décodés par un 1 logique et les symboles « étroits » sont décodés par un 0 logique.

Algorithme 5: Encodage des données dérobées par le processus espion

```
Input: adresse, donnée_à_transférer
  For i = donnée_à_transférer_size To 0 Do
    If (donnée_à_transférer[i] = 1) Then
      Expulser_ligne_cache (adresse)
      loop for Tempo_1
    Else
      Expulser_ligne_cache (adresse)
      loop for Tempo_2
    End If
    Accès(adresse)
    loop for Tempo_3
  End For
```

Algorithme 6 : Décodage des données dérobées par le processus récepteurs

```
Input: adresse
Output: donnée
  While(1)
    Temps = Measure_clock_cycles (Accès(adresse))
    i++
    Exploitation des données dérobées
    If Temps > seuil Then
        donnée[i] = 1
    Else
        donnée[i] = 0
    End If
```

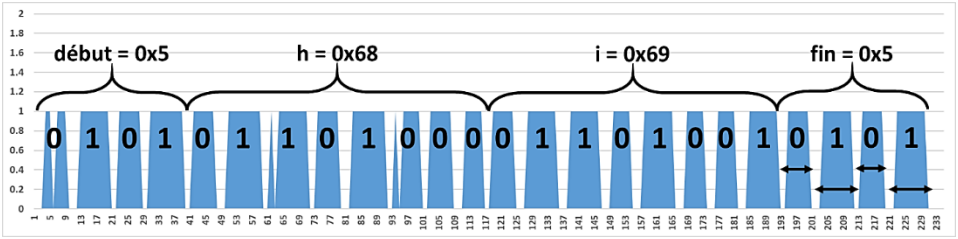

Figure 45: le décodage de l'information dérobée

L'attaque présentée montre la possibilité d'une attaque par canal dérobé par exploitation malicieuse de la cohérence de cache entre une application logicielle malicieuse exécutée dans le monde sécurisé de la partie processeur du SoC et une IP matérielle corrompue (par un cheval de Troie) de la partie reconfigurable du SoC. Pour transmettre une information de taille importante, un protocole simple est utilisé : l'envoie de quatre bits spécifiques pour indiquer le début de la transmission et de quatre bits spécifiques pour indiquer la fin de la transmission. Ce protocole

peut être développé par exemple, par l'ajout d'information telle que la taille de la trame échangée par exemple.

Enfin, la figure 47 montre l'effet de la politique de remplacement de la ligne de cache sur la bande passante du canal dérobé. La politique de remplacement round-robin prend plus de temps pour remplacer une ligne de la mémoire cache ce qui augmente le temps de transmission d'une trame de données et diminue ainsi la bande passante.

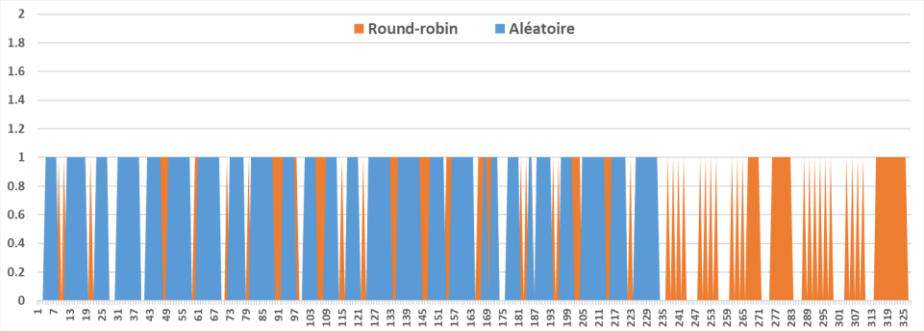

Figure 46: L'impact de la politique de remplacement sur la bande passante

4. Conclusion :

Ce chapitre a apporté la preuve que le mécanisme de cohérence de la mémoire cache dans les SoC complexes hétérogènes constitue une grande menace pour un système sécurisé. Il a notamment montré que les attaques classiques par analyse temporelle de la mémoire cache (telles que les attaques Flush+Reload et Evict+Time) étaient possibles à réaliser depuis la partie reconfigurable. Ces attaques pouvant, comme cela a était montré dans ce chapitre, cibler l'exécution d'un algorithme de chiffrement symétrique dans le monde sécurisé de la partie processeur du SoC. De plus, ce chapitre a montré que le mécanisme de cohérence de cache pouvait être exploité malicieusement pour mettre en place un canal dérobé dans le SoC entre la partie processeur et la partie reconfigurable et échanger ainsi secrètement et en dehors de toutes les règles de sécurité des informations sensibles entre des processus évoluant dans des mondes différents.