Mise en œuvre d'une attaque par analyse de la consommation de puissance d'un circuit intégré de chiffrement AES et implémentation d'une contre-mesure.

Ing. T. ANIZET
Ir N. MARCHAND
PhD. L. LERMAN
Ir F. DURVAUX
ECAM – Bruxelles

Résumé : Cet article traite en premier lieu de la mise en place d'une attaque par canaux auxiliaires analysant la consommation de puissance d'un circuit intégré de chiffrement AES. Ensuite, on analyse et on traite le développement d'une contre-mesure dont le rôle est d'empêcher l'aboutissement d'attaques ultérieures.

Mots-clefs : attaque par canaux auxiliaires, consommation de puissance, circuit intégré, AES, contre-mesure.

Abstract: This paper details first the implementation of side channel attacks analysing the power consumption of an AES encryption integrated circuit. Then, in this paper we analyze and detail the development of a countermeasure whose role is to prevent the successful completion of subsequent attacks.

Keywords: side channel attacks, power consumption, integrated circuit, AES, countermeasure.

1. Contexte

1.1. *Advanced Encryption Standard*

En 1997, le NIST[1] décida de développer un nouveau standard d'algorithme de chiffrement : l'algorithme **AES**[2]. Ce nouveau standard était appelé à remplacer l'ancien standard de chiffrement, l'algorithme DES[3]. L'AES est un **algorithme de chiffrement symétrique par blocs**. La notion de *blocs* signifie que les données sont traitées par blocs de 128 bits. La clé secrète peut posséder différentes tailles : 128 bits (AES-128), 192 bits (AES-192) ou encore 256 bits (AES-256). L'algorithme AES est également caractérisé par une série de tours (*rounds*) dépendant de la taille de la clé (voir tableau 1). Un *round* est lui-même défini par 4 opérations[4] appliquées successivement sur une matrice de données. Ces 4 opérations [1] sont : *AddRoundKey*, *SubBytes*, *ShiftRows* et *MixColumns*. La figure 1 permet de visualiser l'ordre d'exécution chronologique de ces 4 opérations. Une cinquième opération est également définie : l'opération *KeySchedule* [1]. Dans cet article, il est question de réaliser une attaque CPA[5] sur l'algorithme **AES-256**.

	Taille de la clé (bits)	Taille du bloc de données (bits)	Nombre de *rounds*
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Tableau 1 - Principales caractéristiques des 3 variantes de l'algorithme AES.

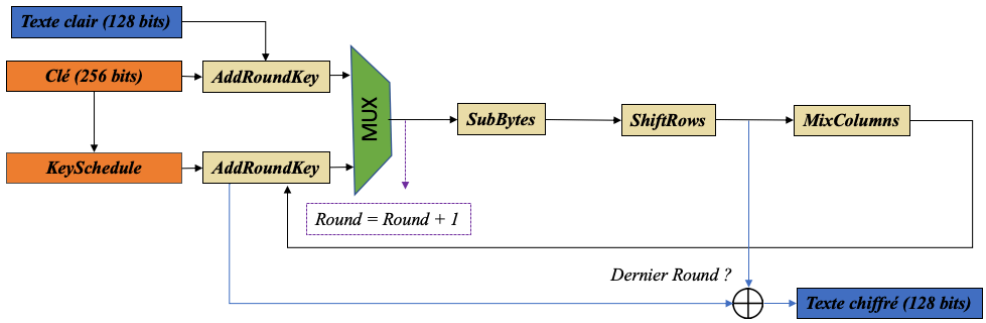

Figure 1 - Fonctionnement général de l'algorithme AES.

[1] *National Institute of Standards and Technology*
[2] *Advanced Encryption Standard*
[3] *Data Encryption Standard*
[4] À l'exception du dernier round qui ne compte que 3 opérations : *SubBytes, ShiftRows* et *AddRoundKey.*
[5] *Correlation Power Analysis*

1.2. Consommation de puissance des circuits en technologie CMOS

La **technologie CMOS**[6] est la technologie la plus répandue parmi toutes les technologies de semi-conducteurs. Le circuit analysé dans ce travail utilise cette technologie : il s'agit d'un FPGA[7] de la famille Spartan 6 sur lequel est implémenté l'algorithme AES-256. Il est évident que les circuits digitaux modernes consomment de la puissance lorsqu'ils exécutent des instructions. Dans le domaine de la cryptanalyse, cette puissance est mesurée et analysée afin de déterminer si le système cryptographique (FPGA) laisse fuiter des informations. Si tel est le cas, ces fuites d'informations peuvent être utilisées afin de retrouver la clé de chiffrement[8] de l'algorithme AES.

La **consommation totale de puissance** d'un circuit en technologie CMOS peut être obtenue en sommant les consommations de puissance respectives de chaque cellule logique[9] du circuit CMOS. En réalité, la consommation de puissance d'un tel circuit se scinde en deux composantes : la puissance statique et dynamique (voir tableau 2).

- La consommation de ***puissance statique*** des circuits CMOS correspond à la puissance qui est consommée par le circuit lorsqu'il n'y a pas de commutation dans une cellule. Autrement dit, cette puissance est requise pour garder le device en fonctionnement continu. Elle est typiquement très faible (valeur de 80 μW mesurée dans le cadre de ce travail – voir figure 2) et sera, en pratique, négligée.
- La consommation de ***puissance dynamique*** des circuits CMOS correspond à la puissance qui est consommée par le circuit lorsqu'il y a une commutation dans la cellule. Autrement dit, cette puissance dépend des opérations exécutées et des données manipulées. Elle constitue un facteur dominant dans la consommation de puissance totale (valeur de 3450 μW mesurée dans le cadre de ce travail – voir figure 2). Il est donc primordial de pouvoir la calculer [2].

Transitions	***Type de puissance consommée***
0 → 0	Statique
0 → 1	Statique + dynamique
1 → 0	Statique + dynamique
1 → 1	Statique

Tableau 2 - Type de puissance consommée par une cellule CMOS en fonction des 4 transitions d'état de sa sortie.

[6] *Complementary Metal Oxide Semiconductor*
[7] *Field-Programmable Gate Array*
[8] D'autres informations peuvent également être retrouvées.
[9] Une cellule logique est conçue pour remplir une certaine fonction logique.

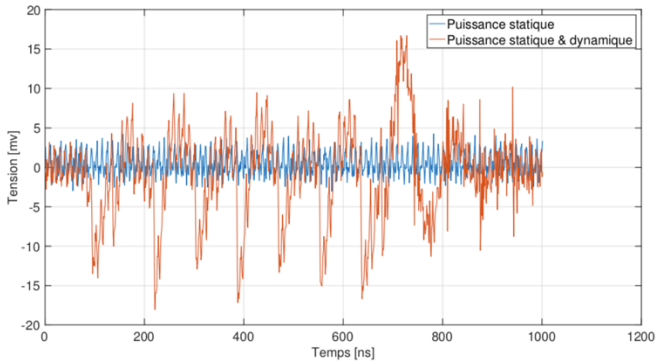

Figure 2 - Comparaison de la "puissance" statique et dynamique.

Pour mesurer la consommation de puissance du circuit intégré, l'attaquant utilise un instrument de mesure. Celui utilisé pour ce travail est l'oscilloscope. Ainsi, l'oscilloscope permet de capturer et d'enregistrer des données, appelées ***traces***, mesurées à partir des canaux auxiliaires du circuit électronique. Pour réaliser la mesure à l'oscilloscope, une résistance est placée en série avec le canal (la PIN) connecté à la tension d'alimentation V_{DD} du système cryptographique. L'oscilloscope est alors en mesure d'enregistrer une différence de potentiel $V(t)$ (en AC) aux bornes de la résistance. Étant donné que le courant $I_{DD}(t)$ circulant dans le système cryptographique est de très faible valeur (μA), la tension $V(t)$ est également très faible. Un amplificateur est généralement utilisé afin d'amplifier cette différence de potentiel. La figure 3 ci-dessous présente le principe de mesure à l'oscilloscope.

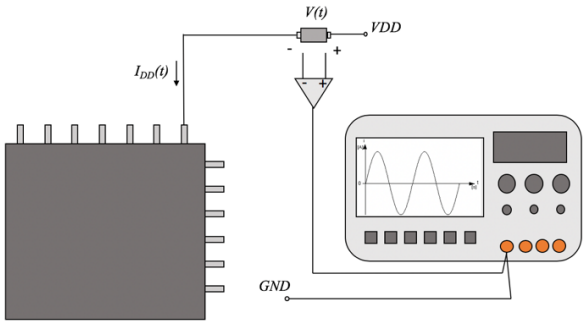

Figure 3 - Principe de mesure à l'oscilloscope.

Remarque : On parle d'attaque par analyse de la consommation de puissance or avec un oscilloscope, on mesure une tension et non une puissance. Comme le démontre l'équation *(2)*, la puissance consommée $p(t)$ est proportionnelle à la tension mesurée $V(t)$. Il ne s'agit donc pas d'une grossière erreur de parler de consommation de puissance même s'il faudrait parler de tension pour être précis. En supposant que la tension d'alimentation V_{DD} est constante et par simple application de la loi d'Ohm, on a les deux relations suivantes (1) :

$$\begin{cases} u(t) = V_{DD} \\ i(t) = \dfrac{V(t)}{R} \end{cases} \quad (1)$$

En reprenant la définition de la puissance consommée et en y remplaçant les termes $u(t)$ et $i(t)$, on obtient (2) :

$$p(t) = u(t).i(t) = V_{DD}.\frac{V(t)}{R} \quad (2)$$

1.3. Modèles de puissance

Lors d'une attaque par analyse de la consommation de puissance, l'attaquant doit utiliser ce que l'on appelle un **modèle de puissance** [3] afin de prédire la consommation de puissance du système cryptographique attaqué. Différents modèles de prédictions existent. Chacun de ces modèles se base sur les valeurs de bits dans des ensembles de données. La qualité du modèle employé a un impact important sur l'efficacité de l'attaque. Deux modèles sont généralement définis et utilisés : Il s'agit des modèles de **Poids de Hamming** (*Hamming Weight* - HW) et de **Distance de Hamming** (*Hamming Distance* - HD). Ces deux modèles restent généraux, c'est-à-dire qu'ils ne requièrent pratiquement aucune connaissance à propos du *design* du circuit et sont par conséquent parfois imprécis.

1.3.1. Modèle de poids de Hamming

Le poids de Hamming est le modèle de consommation de puissance le plus élémentaire. C'est celui le plus utilisé par un attaquant lorsqu'il s'agit d'estimer la consommation de puissance d'un circuit dont on ne connait pas certaines valeurs intermédiaires consécutives calculées durant l'exécution de l'algorithme. Ce modèle considère qu'un bit à 0 ne mène à aucune quantité significative de consommation de puissance au contraire d'un bit à 1. Ainsi, pour ce modèle, on suppose que la consommation de puissance prédite est proportionnelle au nombre de bits à 1 d'une donnée traitée. soit traduit mathématiquement (3) :

$$W(b_1) = \sum_{i=0}^{N-1} b_{1,i} \quad (3)$$

Où b_1 est un mot de N bits. Par conséquent, $b_{1,i}$ est le i^{ème} bit de mot binaire b_1.
Exemple : *HW(*100110*)* = 3

1.3.2. Modèle de distance de Hamming

La distance de Hamming est un modèle de consommation de puissance proposé par Brier, Clavier et Olivier [4]. Il est basé sur la relation entre la consommation de puissance et l'activité de commutation dans les circuits en technologie CMOS. Ce modèle suppose que la puissance totale consommée par un circuit est équivalente à la consommation de puissance lors de commutations (transitions 0 à 1 et 1 à 0). La distance calculée sur base de ce modèle est donc proportionnelle au nombre de transitions 0 à 1 et 1 à 0. La distance de Hamming entre 2 nombres binaires, b_1 et b_2, se calcule en comptant le nombre de transitions entre ces 2 nombres, soit (4) :

$$HD(b_1, b_2) = HW(b_1 \oplus b_2) \qquad (4)$$

Exemple : *HD(*110110,100100*) = HW(*110110 ⊕ 100100*)* = 2

1.4. Attaque CPA

1.4.1. Définition

Fin des années 90, une nouvelle contrainte pour la conception de systèmes informatiques a vu le jour : la sécurité matérielle. Cette contrainte est apparue du fait que de nouvelles attaques se sont développées : les attaques physiques. Ces attaques exploitent différents types d'information présents physiquement sur le circuit attaqué tels que la consommation de puissance, le rayonnement électromagnétique, le temps de calcul, la température, etc. En effet, les fonctions cryptographiques, bien que pouvant être extrêmement robustes théoriquement (c'est-à-dire mathématiquement) sont très sensibles aux fuites d'informations. C'est-à-dire qu'une quantité très faible d'informations peut être exploitée pour retrouver la clé[10] d'un algorithme cryptographique très fort. Il faut bien insister sur le fait que l'attaque exploite les faiblesses physiques du système cryptographique, c'est-à-dire du système implémentant l'algorithme de chiffrement (AES). En aucun cas, l'attaque n'exploite les failles des principes mathématiques mis en place.

Il existe différents types d'attaques physiques dépendant des paramètres physiques analysés et de la façon dont le circuit est étudié. Comme déjà précisé, ce travail se concentre sur les attaques par **analyse de la consommation de puissance**. Le paramètre physique analysé est donc la consommation de puissance du circuit intégré (FPGA). Il existe également plusieurs types d'attaques analysant la consommation de puissance pour retrouver des informations confidentielles. Ce travail se focalise sur **l'analyse de la consommation de puissance par *corrélation*** (attaque CPA).

[10] D'autres informations peuvent également être retrouvées.

L'attaque par analyse de la consommation de puissance par corrélation [4] est une attaque qui se base sur un calcul de corrélation entre la consommation de puissance du système cryptographique (les traces mesurées à l'oscilloscope) et un modèle de prédiction de puissance (exemple : poids de Hamming). Par définition, le coefficient de corrélation de *Pearson* est un coefficient statistique permettant de mettre en évidence une liaison entre deux types de séries de données statistiques. La valeur du coefficient de corrélation est toujours comprise entre -1 et 1. Cette valeur se calcule de la façon suivante *(5)* :

$$r(X;Y) = \frac{\sum(X-\bar{X}).(Y-\bar{Y})}{\sqrt{\sum(X-\bar{X})^2}.\sqrt{\sum(Y-\bar{Y})^2}} \qquad (5)$$

Où :

- X et Y sont deux séries de données statistiques.
- $\bar{X}$ et $\bar{Y}$ sont les moyennes respectives des variables X et Y.

Sur base de la valeur du coefficient de corrélation obtenue, on peut conclure que :

- Si la valeur absolue du coefficient de corrélation est élevée (proche de 1), il y a une forte liaison entre les deux séries analysées.
- Si la valeur absolue du coefficient de corrélation est faible (proche de 0), il y a une très faible liaison, voir aucun lien entre les deux séries analysées.

Quatre étapes sont nécessaires à l'attaquant pour mettre en place une attaque CPA :

1. **Mesurer la consommation de puissance** : l'attaquant doit mesurer et enregistrer des traces de puissance capturées sur le système cryptographique lorsque celui-ci est en cours de chiffrement (ou de déchiffrement).
2. **Calculer les valeurs intermédiaires hypothétiques** : lorsque l'algorithme AES chiffre des données, des valeurs intermédiaires sont calculées en fonction des opérations exécutées et des données manipulées. L'attaquant doit calculer chacune de ces valeurs intermédiaires. Exemple avec l'algorithme AES-256 : La clé utilisée a une taille de 256 bits. Si l'attaquant recherche un seule byte (8 bits) de la clé, il existe 2^8 soit 256 valeurs de clé possibles. Chacune de ces 256 valeurs de clé va conduire à des résultats intermédiaires différents, fonction des opérations exécutées par l'algorithme, et qui sont appelés *valeurs intermédiaires hypothétiques*.
3. **Utiliser un modèle de puissance** : Il faut ensuite faire correspondre à ces valeurs intermédiaires hypothétiques des valeurs de consommation de puissance. Pour ce faire, l'attaquant utilise un modèle de puissance. Il va ainsi obtenir une prédiction de puissance pour chaque clé testée.
4. **Mesurer la similarité** : Enfin, il faut mesurer la similarité entre les vraies traces de puissance (obtenues au point 1) et les prédictions de traces de puissance (obtenues au point 3). Pour ce faire, l'attaquant calcule le *coefficient de corrélation de Pearson* entre les valeurs estimées et chaque instant sur les vraies traces de puissance. Le résultat de corrélation le plus élevé (en valeur absolue) permettra de retrouver la vraie valeur de la clé.

La figure 4 présente le principe de fonctionnement d'une attaque CPA selon les quatre étapes définies précédemment.

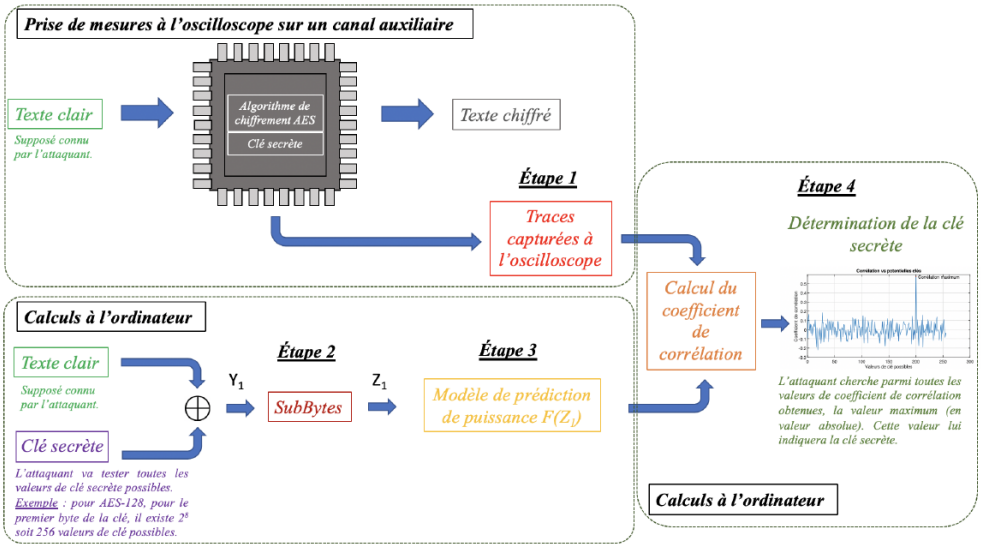

Figure 4 – Principe de fonctionnement d'une attaque CPA.

1.5. Contre-mesure

1.5.1. Définition

Dès que les attaques par analyse de la consommation de puissance ont été reconnues fonctionnelles, une série de contre-mesures [3] a été développée afin d'empêcher la réussite de celles-ci. Les attaques CPA fonctionnent du fait que la consommation de puissance du système cryptographique dépend des valeurs intermédiaires exécutées par l'algorithme de chiffrement. Ainsi, l'objectif d'une contre-mesure est de casser la relation qui existe entre la consommation de puissance et les données sensibles manipulées et entre la consommation de puissance et les opérations exécutées. La contre-mesure développée pour ce travail porte le nom de contre-mesure ***Faking*** [5]. L'objectif de la contre-mesure est le suivant : Dès le départ, on va venir masquer la vraie valeur de clé (Key_{Real}) en opérant un XOR avec un masque (Key_{Mask}) produisant ainsi une fausse clé (Key_{Fake}). On a donc la relation *(5)* :

$$Key_{Fake} = Key_{Real} \oplus Key_{Mask} \qquad (5)$$

À partir de cette fausse clé, on va exécuter l'algorithme AES de façon ordinaire, en appliquant tout d'abord l'opération *KeySchedule* sur la fausse clé de façon à générer d'autres fausses clés puis ensuite exécuter les quatre opérations élémentaires

(*AddRoundKey, SubBytes, ShiftRows, MixColumns*) afin de chiffrer les données. L'implémentation de cette contre-mesure est présentée à la figure 5. C'est bien la fausse clé qui est utilisée pour chiffrer les données et non la vraie clé. De cette manière, en supposant que l'attaquant ne connaisse pas la valeur du masque (Key_{Mask}) appliqué sur la vraie clé, si ce dernier décide d'opérer une attaque CPA sur le système, il sera capable de retrouver une valeur de clé. Cependant, il s'agira de la fausse clé générée dès le départ.

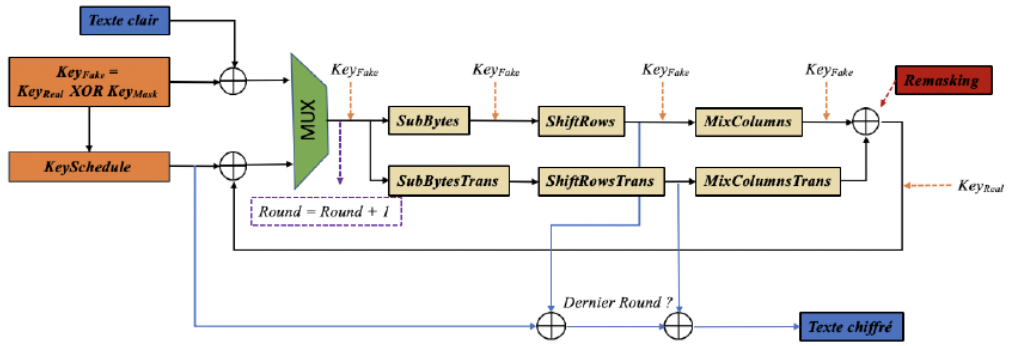

Figure 5 - Implémentation de la contre-mesure faking pour l'algorithme de chiffrement AES-256.

Pour toute contre-mesure implémentée, lorsqu'un masque est appliqué sur les données intermédiaires, il faut ensuite le retirer de façon à obtenir *in fine* le *vrai* texte chiffré. Si aucune action supplémentaire n'était entreprise, le texte chiffré serait chiffré avec la fausse clé plutôt qu'avec la vraie clé. Pour ce faire, trois nouvelles opérations sont ajoutées. Leur rôle est d'annuler l'emploi du masque sur la clé. Ces trois opérations sont : *SubBytesTrans*, *ShiftRowsTrans* et *MixColumnsTrans*. Enfin, dans le but de couper définitivement tout lien des résultats intermédiaires avec la fausse clé, un XOR (opération *remasking*) est opéré entre les résultats intermédiaires obtenus après *MixColumns* et ceux obtenus après *MixColumnsTrans*. Le résultat de ce XOR permet de retirer l'emploi du masque (Key_{Mask}). On retrouve alors le résultat que l'on aurait obtenu en sortie de l'opération *MixColumns* si on avait utilisé la vraie clé initialement. Ceci est visible sur la figure 5. Ensuite, l'opération *AddRoundKey* emploie à nouveau une fausse clé, dérivant de l'opération *KeySchedule*.

Voici la définition de la nouvelle opération *SubBytesTrans*. Notons $Key_{Fake}(i,j)$ (avec $i \in [0:3]$ et $j \in [0:3]$) la fausse clé utilisée, $Key_{Real}(i,j)$ la vraie clé utilisée et $P(i,j)$ l'ensemble de textes clairs envoyés au système cryptographique. Notons également $a_F(i,j)$ un byte particulier à la sortie de l'opération *AddRoundKey* chiffré avec la fausse clé $Key_{Fake}(i,j)$ et notons $a_R(i,j)$ un byte particulier à la sortie de l'opération *AddRoundKey* mais cette fois-ci chiffré avec la vraie clé Key_{Real}. Ces deux notations sont reprises à l'équation *(6)*.

$$\begin{cases} a_F(i,j) = Key_{Fake}(i,j) \oplus P(i,j) \\ a_R(i,j) = Key_{Real}(i,j) \oplus P(i,j) \end{cases} \quad (6)$$

Étant donné que c'est la fausse clé qui est utilisée pour l'implémentation de la contre-mesure, si nous souhaitons obtenir le résultat intermédiaire à la sortie de l'opération *SubBytes*, nous avons la relation *(7)* :

$$Sbox\big(a_F(i,j)\big) = Sbox(Key_{Fake}(i,j) \oplus P(i,j)\,) \quad (7)$$

Connaissant la relation *(5)*, nous pouvons réécrire l'équation *(7)* sous la forme suivante *(8)* :

$$Sbox\big(a_F(i,j)\big) = Sbox(Key_{Real}(i,j) \oplus Key_{Mask}(i,j) \oplus P(i,j)\,) \quad (8)$$

Ainsi, une façon simple de retrouver le byte original $a_R(i,j)$ utilisé avec la clé réelle à la sortie de l'opération *SubBytes* est de définir une nouvelle fonction, appelée *SubBytesTrans*. Cette fonction est donc définie sur base de l'équation *(8)* de la manière suivante :

$$Sbox\big(a_F(i,j)\big) = Sbox(Key_{Real}(i,j) \oplus P(i,j)\,) \oplus SubBytesTrans(a_F(i,j)) \quad (9)$$

En effet, selon l'équation *(6)*, on peut simplifier l'équation précédente *(9)* par l'équation suivante *(10)* où l'on retrouve bien le byte original $a_R(i,j)$.

$$Sbox\big(a_F(i,j)\big) = Sbox(a_R(i,j)) \oplus SubBytesTrans(a_F(i,j)) \quad (10)$$

Autrement dit, la fonction *SubBytesTrans* se traduit par :

$$SubBytesTrans\big(a_F(i,j)\big) = Sbox\big(a_R(i,j)\big) \oplus Sbox(a_F(i,j)) \quad (11)$$

Concernant les opérations *ShiftRowsTrans* et *MixColumnsTrans*, il ne s'agit en réalité pas de nouvelles opérations étant donné que ces deux opérations réalisent respectivement la même fonction que les opérations *ShiftRows* et *MixColumns*. Cependant, il faut préciser que le fait d'exécuter ces deux opérations permet d'éviter une faille dans la contre-mesure (voir section 1.5.2.). C'est donc la raison pour laquelle, le *remasking* (opération XOR) ne se fait pas avant cette opération.

1.5.2. Points faibles et améliorations

Deux points faibles [5] de l'implémentation de la contre-mesure faking proposés à la figure 5 sont repris ci-dessous et améliorés afin de diminuer grandement les chances de réussite de l'attaquant.

(1) Le but de la contre-mesure faking est d'autoriser l'attaquant à trouver une fausse valeur de clé. Cependant, connaissant la valeur de cette faussé clé, si l'attaquant trouve la valeur du masque, il peut retrouver la valeur de la vraie clé selon l'équation *(5)*. Pour diminuer les chances de l'attaquant de retrouver la valeur de la vraie clé connaissant la fausse clé, chaque valeur du byte du masque utilisé (Key_{Mask}) est choisie de manière uniformément aléatoire.

(2) Il existe une attaque utilisant les résultats intermédiaires à la sortie des opérations *SubBytes* et *SubBytesTrans* et qui permet de retrouver la clé réelle. En effet, si l'attaquant récupère ces deux valeurs intermédiaires et qu'il réalise un XOR entre elles, il obtient la simplification suivante *(12)* :

$$Sbox\big(a_F(i,j)\big) \oplus SubBytesTrans\big(a_F(i,j)\big) = Sbox\big(a_R(i,j)\big) \quad (12)$$
$$= Sbox\big(Key_{Real}(i,j)\big) \oplus State(i,j)$$

En considérant le premier round de l'AES-256, l'élément $State(i,j)$ représente un texte clair $P(i,j)$. Dans ce cas, connaissant le fonctionnement de l'opérations *SubBytes*, l'attaquant peut retrouver la valeur de la vraie clé Key_{Real}. Pour se prémunir de ce problème, une amélioration de la contre-mesure faking consiste à masquer les valeurs en sortie de l'opération *SubBytesTrans*. En effet, à chaque round, un masque aléatoire $M_h(i,j)$ est appliqué sur les résultats intermédiaires en sortie de l'opération *SubBytesTrans* de sorte à obtenir l'équation suivante *(13)* :

$$SubBytesTrans\big(a_F(i,j)\big) = Sbox\big(a_R(i,j)\big) \oplus Sbox\big(a_F(i,j)\big) \oplus M_h(i,j) \quad (13)$$

Notons que le masque $M_h(i,j)$ est renouvelé de façon aléatoire à chaque round pour complexifier le chiffrement. Ce masque $M_h(i,j)$ devient le masque $M_s(i,j)$ après l'exécution de l'opération *ShiftRowsTrans* et devient ensuite le masque $M_k(i,j)$ après l'exécution de l'opération *MixColumnsTrans*. La figure 6 présente le principe de fonctionnement de la contre-mesure faking avec les améliorations apportées.

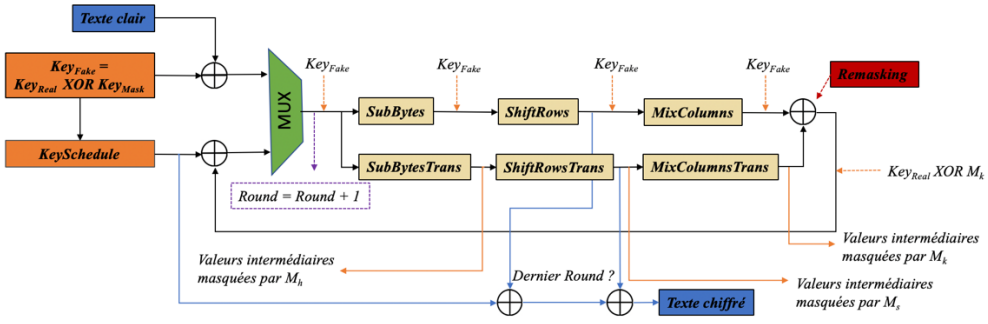

Figure 6 - Implémentation de la contre-mesure faking pour l'algorithme de chiffrement AES-256 avec prise en compte des améliorations.

2. Résultats de l'attaque CPA sur l'implémentation non protégée de l'AES-256

2.1. Traces brutes

La figure 7 présente une trace brute[11] capturée à l'oscilloscope. La tension consommée par le FPGA varie entre −104 mV et 56 mV. L'algorithme met approximativement 1500 ns pour chiffrer un message clair.

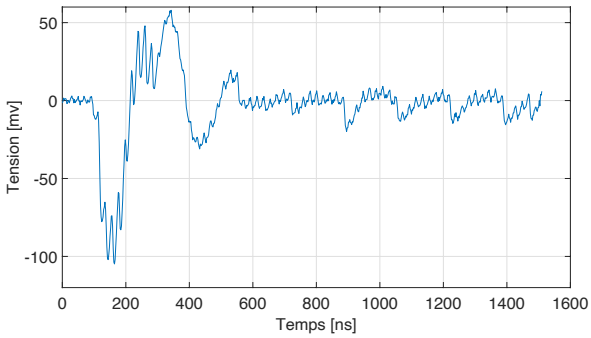

Figure 7 - Trace brute.

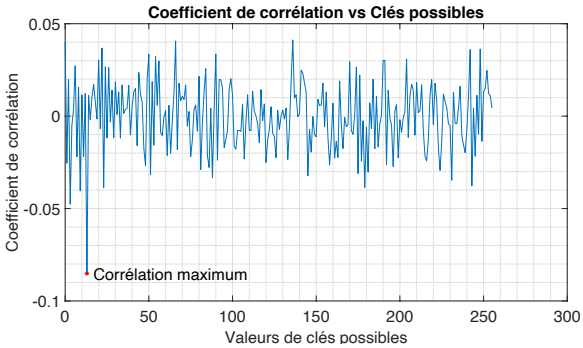

Figure 8 - Résultat de l'attaque CPA sur le quatorzième byte de la clé.

La figure 8 présente le résultat d'une attaque CPA sur le quatorzième byte de la clé de chiffrement de l'algorithme AES-256. 5000 textes clairs[12] ont été utilisés pour obtenir ce résultat. La valeur maximale de corrélation vaut -0,085 et révèle la valeur décimale 13 pour le quatorzième byte de la clé secrète (correct). La procédure permettant d'obtenir ce type de graphe est appliquée sur les trente-et-un bytes restants de la clé afin d'obtenir au final l'entièreté de la clé secrète de l'AES.

La figure 9 représente le résultat d'une attaque CPA où l'on fait varier le nombre de traces pour retrouver la valeur de la clé. Cette attaque est réalisée sur un byte de la clé. La figure 9 présente donc 256 courbes de corrélation qui varient selon le nombre de traces utilisées et dont une représente la vraie clé employée par l'algorithme AES-256 (courbe verte). Ainsi, la figure 9 permet de déduire le nombre de traces minimum qu'un attaquant doit utiliser pour que l'attaque CPA révèle avec certitude la clé de chiffrement. Il s'agit encore de résultats obtenus par rapport au quatorzième byte de la clé. Dans le cas présent de cette attaque, au-dessus de 1400 traces, l'attaque CPA révèle toujours la vraie valeur de la clé secrète. De plus, au plus le nombre de traces augmente, au plus la clé secrète se distingue des 255 autres clés testées par l'attaquant.

---

[11] Pas de prétraitement ou de post-traitement appliqué sur la trace.

[12] Générés aléatoirement à chaque exécution de l'AES. (5000 textes clairs = 5000 traces de puissance)

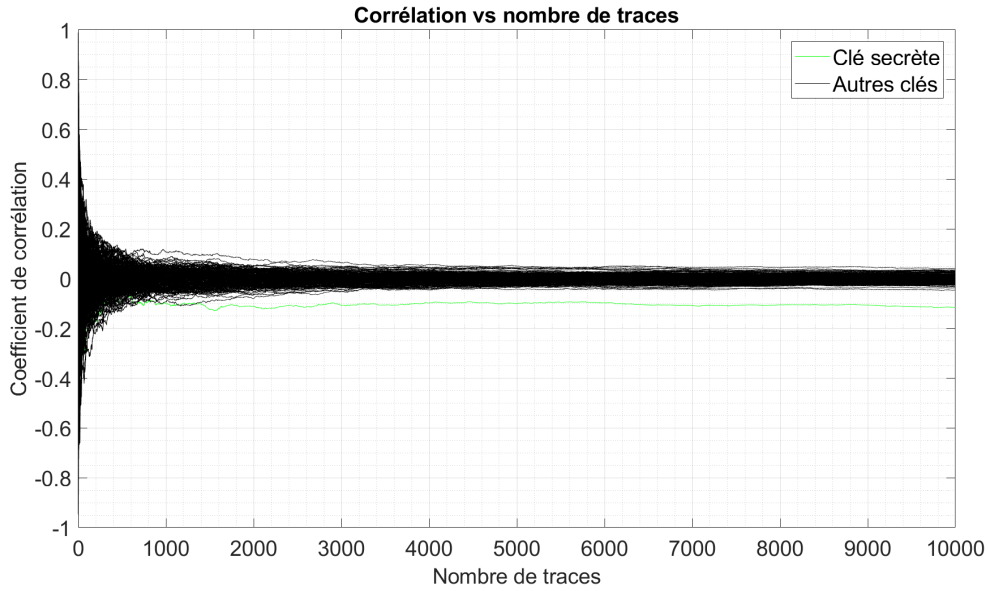

Figure 9 - Attaque CPA sur le quatorzième byte de la clé de chiffrement. Il faut un minimum de 1400 traces pour que l'attaque CPA fonctionne tout le temps.

2.2. Traces avec prétraitement

Le bruit a une influence capitale en traitement de signaux. De ce fait, un prétraitement des traces peut être opéré afin de retirer le bruit parasite inutile. Une façon efficace d'enlever ce bruit parasite est d'utiliser un filtre. L'utilisation de ce filtre a pour objectif d'augmenter le taux de succès de l'attaque. Sachant que le Picoscope possède une bande passante de 200 MHz et qu'il échantillonne à 500 Mé/s (500 MHz) pour une résolution sur 12 bits et sachant que la fréquence d'horloge utilisée par le FPGA est de 48 MHz, l'idée consiste à utiliser un filtre numérique de type passe-bas. Plus précisément, le filtre employé est un filtre à réponse impulsionnelle finie (*FIR*) d'ordre 100 et dont la fréquence de coupure est définie à 48 MHz. La fenêtre employée est une fenêtre de *Hamming*. La figure 10 compare une trace brute à une trace filtrée. L'application du filtre a pour effet de *lisser* la trace. Cependant, quand il s'agit d'analyser le traitement d'un signal, il est plus commode d'observer ce signal dans le domaine fréquentiel. Ainsi, les figures 11 et 12 présentent respectivement le résultat de la transformée de Fourier discrète sur une trace brute et sur une trace filtrée. L'algorithme *FFT*[13] a été utilisé pour calculer cette transformée de Fourier discrète. La figure 12 permet de conclure que le bruit parasite au-dessus de la fréquence de 48 MHz a bien été enlevé suite à l'application du filtre passe-bas.

[13] *Fast Fourier Transform*

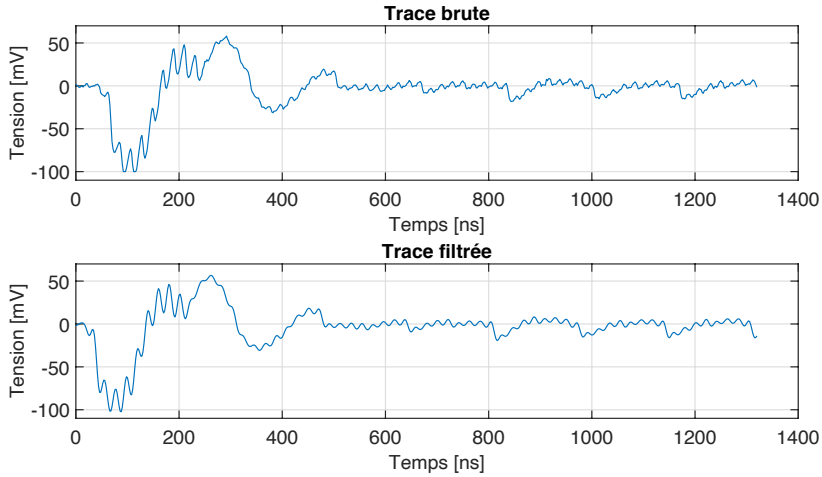

Figure 10 - Comparaison dans le domaine temporel entre une trace brute et une trace filtrée. Le filtre FIR de type passe-bas a pour effet de lisser le signal.

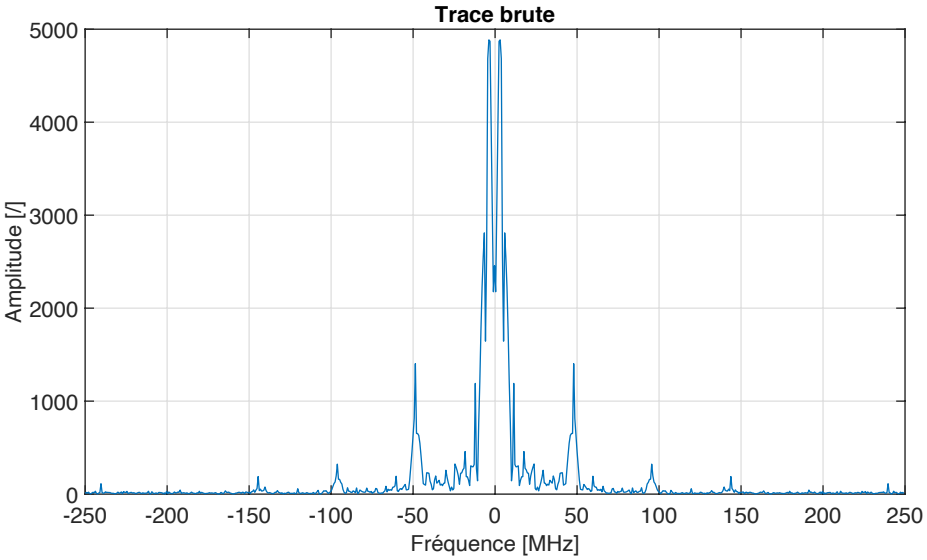

Figure 11 - Transformée de Fourier discrète pour une trace brute.

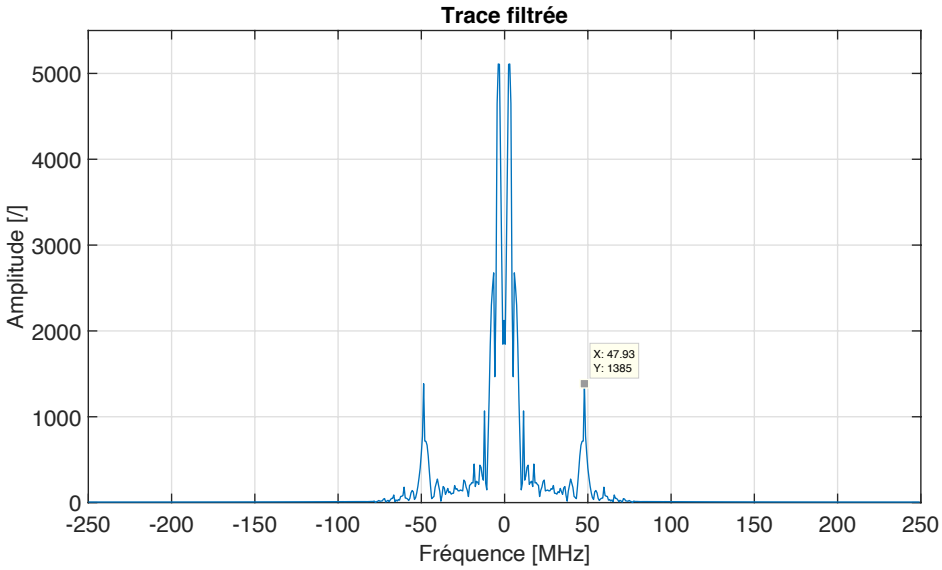

Figure 12 - Transformée de Fourier discrète pour une trace filtrée.

La figure 13 indique le nombre de traces filtrées qu'un attaquant doit utiliser pour que l'attaque CPA révèle la valeur du quatorzième byte de la clé de chiffrement. Dans le cas présent de cette attaque, au-dessus de 1300 traces, la valeur de la clé secrète est toujours retrouvée. On remarque également que la corrélation pour la clé secrète est plus élevée lorsque le filtrage est appliqué (valeur de 0,169 pour 10000 traces) que lorsqu'il ne l'est pas (valeur de 0,118 pour 10000 traces).

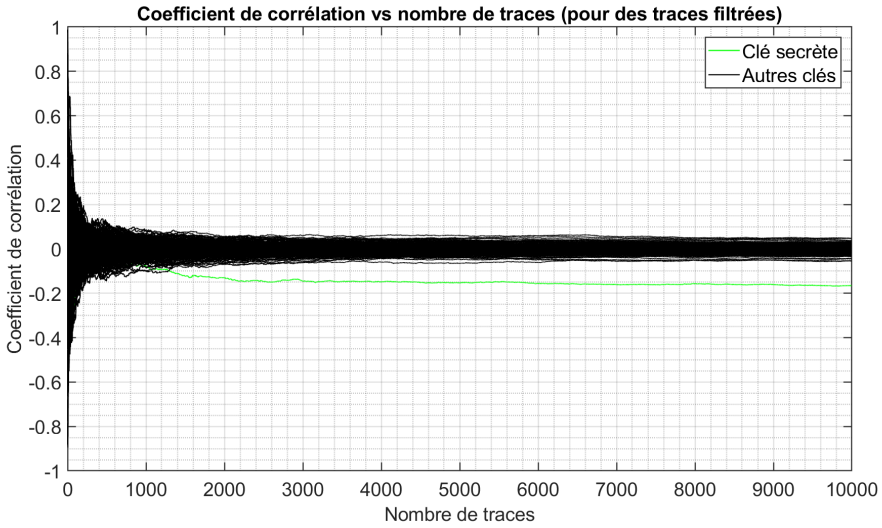

Figure 13 - Attaque CPA sur le quatorzième byte de la clé de chiffrement. Il faut un minimum de 1300 traces pour que l'attaque CPA fonctionne tout le temps.

2.3. Taux de succès de l'attaque

Pour mieux visualiser l'impact du filtrage sur les résultats, une métrique est utilisée afin de mesurer le taux de succès de l'attaque. Le taux de succès de l'attaque représente la probabilité que l'adversaire réussisse à extraire la bonne clé. Ce taux de succès est représenté en fonction du nombre de traces utilisées. La figure 14 présente le taux de succès de l'attaque avec et sans filtrage sur le cinquième byte de la clé secrète. On constate clairement l'avantage qu'apporte le filtrage sur les traces mesurées.

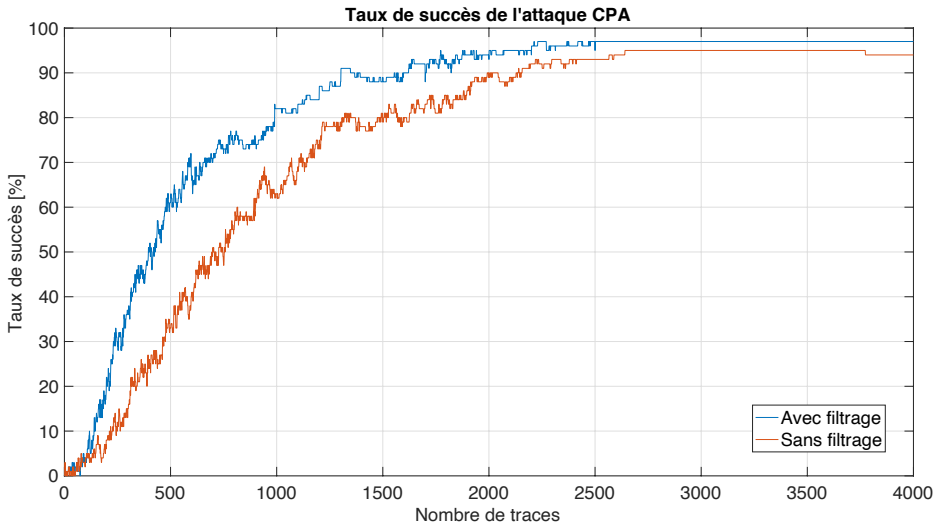

Figure 14 - Taux de succès de l'attaque CPA sur le cinquième byte de la clé selon que les traces soient filtrées (courbe bleue) ou non (courbe orange). L'attaque fonctionne mieux avec filtrage.

3. Développement de la contre-mesure

Deux architectures de la contre-mesure faking ont été implémentées : une architecture en parallèle et une en séquentiel. L'implémentation en séquentiel a été réalisée sur base des résultats[14] obtenus sur l'implémentation en parallèle. L'implémentation en parallèle signifie que les opérations *SubBytes*, *ShiftRows* et *MixColumns* sont respectivement exécutées parallèlement aux opérations *SubBytestrans*, *ShiftRowsTrans* et *MixColumnsTrans*. L'implémentation en séquentiel signifie que toutes les opérations sont exécutées séquentiellement : *SubBytes – ShiftRows – MixColumns – SubByteTrans – ShiftRowsTrans – MixColumnsTrans*.

[14] Détaillés aux sections 3.1. et 3.2.

3.1. Contre-mesure avec implémentation en parallèle

La figure 15 présente le résultat de l'attaque CPA pour le cinquième byte de la clé de chiffrement. La clé révélée (138) ne correspond pas à la vraie clé (4), ce qui est normal puisque la vraie clé est masquée. Cela montre bien que la contre-mesure joue correctement son rôle afin d'empêcher l'attaquant de retrouver la valeur de la clé secrète. Cependant, le résultat obtenu est inattendu. La clé retrouvée ne correspond pas non plus à la fausse clé utilisée (84). Or, en théorie[15], cette fausse clé devrait être retrouvée par l'attaquant. De nouvelles attaques plus ciblées ont été mises en œuvre. Cependant, le résultat reste identique, maximum six bytes de la fausse clé (sur les 32 bytes) peuvent être retrouvés. L'une des raisons pouvant justifier le non-fonctionnement de la contre-mesure concerne l'exécution en parallèle des opérations. Cette exécution en parallèle des opérations pourrait avoir comme conséquence de générer du bruit parasite. Ainsi, l'idée de modifier l'architecture de la contre-mesure en séquentiel a été établie.

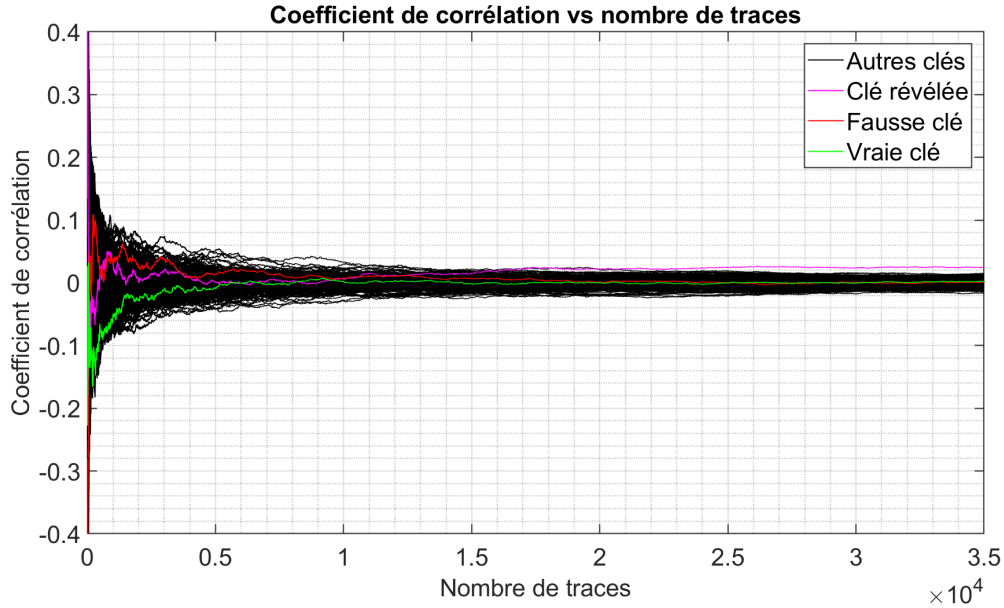

Figure 15 - Valeurs de corrélation pour les 256 clés testées en fonction du nombre de traces mesurées. La vraie clé (courbe verte) n'est pas révélée. La fausse clé (courbe rouge) n'est pas non plus révélée. La clé révélée (courbe magenta) est un résultat inattendu.

[15] Voir équation *(5)*

3.2. Contre-mesure avec implémentation en séquentiel

La figure 16 présente le résultat de l'attaque CPA sur le cinquième byte de la clé de chiffrement. Comme pour l'architecture parallèle, la clé retrouvée (201) ne correspond pas à la vraie clé (4), ce qui est normal puisque la vraie clé est masquée. Cela montre à nouveau que la contre-mesure joue correctement son rôle afin d'empêcher l'attaquant de retrouver la valeur de la clé secrète. Cependant, le résultat obtenu n'est pas plus satisfaisant. La clé retrouvée ne correspond pas non plus à la fausse clé utilisée (84). Plus incompréhensible, après plusieurs attaques, le résultat n'est pas meilleur que celui obtenu lorsque l'architecture est configurée en parallèle : maximum six bytes de la fausse clé peuvent être retrouvés. Dans un article [5] détaillant la contre-mesure *faking*, il est précisé que les opérations exécutées spécifiquement pour la contre-mesure *faking* (*SubBytesTrans, ShiftRowsTrans, MixColumnsTrans* et *Remasking*) sont implémentées sur un second FPGA. Par manque de temps, cela n'a pas pu être testé. Cependant, il s'agit d'une piste d'amélioration à ne pas négliger. Une question reste cependant en suspens : *pour quelles raisons retrouve-t-on dans 75% des attaques CPA réalisées, quatre bytes de la fausse clé de chiffrement ?* Sur base de cette question, deux pistes peuvent être envisagées :

- Mieux cibler l'attaque : lorsque l'attaque CPA est réalisée, elle se concentre sur un certain nombre d'échantillons dans la trace. La mise en œuvre d'une métrique permettant de déduire dans les traces, les échantillons à cibler pour améliorer le taux de succès de l'attaque semble être une première piste afin d'identifier où l'information peut être trouvée.
- Utilisation d'un filtre performant : certes, un filtre *FIR* passe-bas a été mis en place et permet d'obtenir de meilleurs résultats. La bande passante après application de ce filtre s'étend de 1 Hz à 50 MHz. Cette bande passante peut être analysée afin de déterminer sur quelles fréquences les informations relatives à la clé secrète peuvent être révélées. En effet, les informations relatives à la clé sont révélées à basses fréquences.

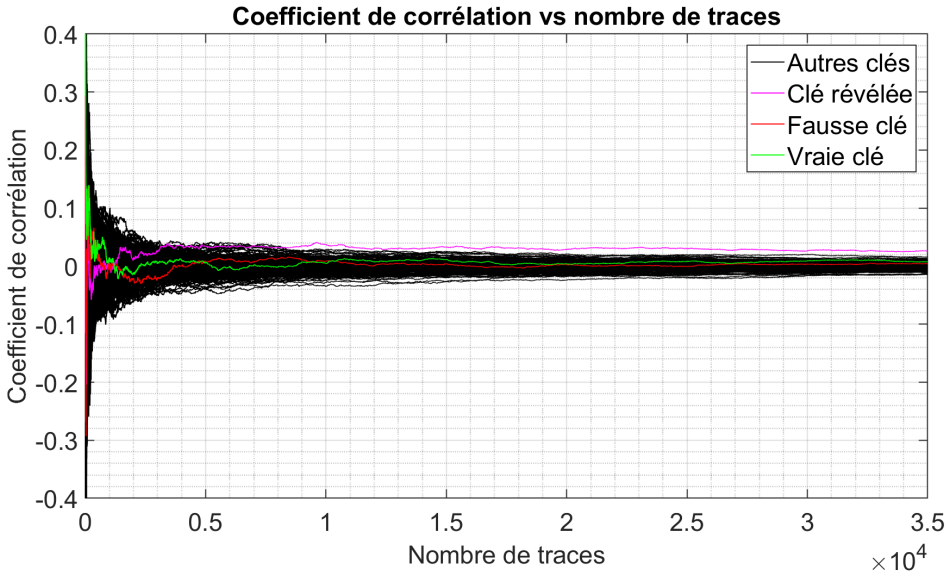

Figure 16 - Valeurs de corrélation pour les 256 clés testées en fonction du nombre de traces mesurées. La vraie clé (courbe verte) n'est pas révélée. La fausse clé (courbe rouge) n'est pas non plus révélée. La clé révélée (courbe magenta) est un résultat inattendu.

4. Conclusion

Cet article décrit la mise en œuvre d'un démonstrateur employé afin de mettre en évidence l'impact d'une attaque par analyse de la consommation de puissance par corrélation sur une implémentation non protégée de l'algorithme AES-256. Le résultat d'une telle attaque sur l'implémentation de l'algorithme AES-256, pourtant réputé robuste et incassable, ne met qu'une quinzaine de minutes pour révéler ses secrets les plus confidentiels. Ce résultat permet de conclure que les procédures développées et mises en place sont correctement configurées. Une amélioration de l'attaque a été mise au point afin d'en augmenter son taux de succès. Cette amélioration consiste à appliquer un filtre sur les traces, ce qui permet d'en extraire toute information inutile avant l'exécution de l'attaque. La métrique du taux de succès de l'attaque CPA permet de conclure qu'il faut seulement 1000 traces pour obtenir un résultat satisfaisant avec filtrage (80% de réussite) alors que 500 traces supplémentaires sont nécessaires pour obtenir ce même résultat sans filtrage. Au final, on obtient donc un démonstrateur performant qui permet de révéler assez rapidement la valeur de la clé secrète employée par l'algorithme AES-256.

La contre-mesure développée et implémentée, nommée faking, remplit correctement sa tâche. Lorsque le démonstrateur est utilisé par un attaquant, la contre-mesure l'empêche de retrouver la valeur de la clé secrète employée par l'algorithme AES-256. La totalité (100%) des attaques réalisées sur l'implémentation de l'algorithme AES-256 avec présence de la contre-mesure faking a révélé une clé de chiffrement différente de celle réellement manipulée par l'algorithme. Par conséquent, l'attaquant est dans l'incapacité de déchiffrer toute donnée confidentielle préalablement chiffrée. Cependant, il persiste un questionnement quant à la clé révélée par l'attaque CPA. Si cette clé ne correspond en aucun cas à la vraie clé, elle ne correspond pas non plus à la fausse clé censée être retrouvée. On observe donc un écart entre les résultats obtenus en pratique et ceux attendus en théorie. Trois pistes d'améliorations futures basées sur les résultats obtenus ont été succinctement proposées et abordées dans cette étude et mériteraient que l'on s'y penche plus longuement afin d'améliorer et enrichir la solution proposée. La première piste consiste à exécuter les opérations spécifiques à la contre-mesure sur un second FPGA. La seconde piste consiste à mieux cibler les échantillons dans la trace de puissance où de l'information peut être exploitée. La dernière piste consiste en l'utilisation d'un filtre performant permettant de déterminer sur quelles fréquences les informations relatives à la clé secrète peuvent être révélées.

Enfin, le tableau 3 rassemble les principales performances analysées pour l'implémentation de l'AES-256 seul, l'AES-256 avec contre-mesure faking en mode parallèle et l'AES-256 avec contre-mesure faking en mode séquentiel. L'ajout de la contre-mesure double le nombre de LUT's et de flip-flops configurés. La contre-mesure implémentée en mode séquentiel double la taille mémoire (BRAM) utilisée comparativement à l'implémentation de l'AES seul. La contre-mesure implémentée en mode parallèle augmente le temps d'exécution à 2392 ns alors que celle implémentée en mode séquentiel augmente le temps d'exécution à 3172 ns.

	Slices	***LUT's (Look Up Table)***	***Flip-Flops (Registre)***	***BRAM (9 Kb)***	***IOB (In/Out Block)***	***Temps d'exécution (ns)***
AES seul	1473 (12%)	2742 (5%)	3526 (3%)	8 (2%)	35 (10%)	1586
AES avec contre-mesure faking en mode parallèle	2822 (24%)	5493 (11%)	6734 (7%)	8 (2%)	35 (10%)	2392
AES avec contre-mesure faking en mode séquentiel	2741 (23%)	4970 (10%)	6609 (7%)	16 (4%)	35 (10%)	3172

Tableau 3 - Tableau récapitulatif des performances observées pour les implémentations de l'AES seul, de l'AES avec contre-mesure faking en mode parallèle et de l'AES avec contre-mesure faking en mode séquentiel.

Depuis ces découvertes à la fin des années 90, la sécurité matérielle a pris une tournure particulière pour les grandes industries. C'est désormais sur un nouvel axe de recherche, s'écartant des sentiers traditionnels, que se porte la problématique de protection de données sensibles. Cet article apporte ainsi aux entreprises modernes un outil fonctionnel et performant, permettant de mettre en évidence la puissance d'une attaque par analyse de la consommation de puissance. De plus, le mise en œuvre d'une contre-mesure qui remplit parfaitement sa tâche primaire, à savoir empêcher de révéler la clé secrète employée pour le chiffrement des données confidentielles a été correctement développée. Cette contre-mesure constitue une piste de développement sérieuse pour les industries souhaitant acquérir des produits qui s'affranchissent de ce type d'attaque.

5. Sources

[1] «Why AES Is Secure,» 1 Janvier 2017. [En ligne]. Available: https://weien.io/posts/crypto/why-aes-is-secure.html#substitution-permutation-network.

[2] L. Giancane, «Side-channel attacks and countermeasure in the design of secure ic's devices for cryptographic applications,» 2011. [En ligne]. Available: http://hdl.handle.net/10805/975.

[3] E. O. a. T. P. S. Mangard, Power analysis attacks : revealing the secrets of smart cards., New York: Springer, 2007.

[4] C. C. a. F. O. E. Brier, Correlation Power Analysis with a Leakage Model, Berlin: Springer, 2004, pp. 16-29.

[5] M. L.-G. a. E. C.-N. R. Lumbiarres-Lopez, «Faking Countermeasure Against Side-Channel Attacks,» 2018.