Télécom Paristech

et

Master Parisien de Recherche en Informatique

Cryptanalyse par canaux auxiliaires : attaques et contre-mesures

Projet de fin d'études

Mélissa Rossi Dabi

Encadrants :
Sonia Belaïd
Renaud Dubois

Rapporteur :
Damien Vergnaud
Responsable Pédagogique :
Pierre Senellart

22 décembre 2017

Remerciements

Je tiens avant tout à remercier mes deux encadrants de stage Sonia Belaïd et Renaud Dubois, ingénieurs chercheurs pour le service cryptologie et composants de Thales Communications & Security. Bénéficier de leurs deux expériences complémentaires a été un privilège pour comprendre clairement le monde des canaux auxiliaires. Ils ont toujours été bienveillants, encourageants et optimistes. Je leur suis aussi très reconnaissante de me donner l'opportunité de revenir pour faire une thèse Cifre au sein de ce laboratoire. Merci à Sonia pour avoir mis en place les démarches nécessaires pour ce projet durant le stage.

Je souhaite remercier toute l'équipe professorale et associée du MPRI pour la qualité des cours dispensés dans ce master. J'adresse particulièrement mes remerciements à ma professeur Anne Canteaut qui m'a recommandé de postuler pour ce laboratoire et a appuyé ma candidature.

Merci également à Pierre Senellart, mon directeur des études à Télécom Paristech qui m'a conseillée et a répondu régulièrement à mes questions tout au long de ce stage.

Mes remerciements vont ensuite vers toute l'équipe d'Eric Garrido qui m'a accueillie dans une ambiance agréable. Faire mon stage de fin d'études dans ce laboratoire a été un plaisir, j'ai pu apprendre beaucoup sur les métiers au sein de Thales. Et, j'ai surtout été confortée dans mon projet de poursuivre dans la recherche.

Merci à l'équipe du DPA Contest à Télécom Paristech pour avoir mis en place cette plateforme publique très utile pour se familiariser avec les attaques sur des systèmes réels, et merci pour avoir évalué rapidement mes attaques.

I would like to sincely thank Colin O'Flynn for his help with the Chipwhisperer and the CHES challenge.

J'aimerais adresser des remerciements aux attaquants du CHES Challenge qui m'ont aidée à comprendre leurs attaques et qui m'ont encouragée à participer au Challenge. Merci à Nabil Hamzi qui est allé récupérer mon prix à Santa Barbara.

Je remercie aussi Matthieu Giraud, stagiaire en même temps que moi, qui a pris le temps de m'expliquer l'architecture GNU/Linux et le langage LaTeX.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillée et relue lors de la rédaction de ce rapport : Sonia, Renaud, Matthieu et ma famille.

Le contexte général

Suite au développement des objets connectés, les systèmes embarquées deviennent de plus en plus sophistiqués. Ils contiennent pour la plupart des informations personnelles comme les données bancaires, biométriques ou de santé. Par souci de confidentialité, ces données sont protégées par des systèmes cryptographiques. Les attaques par canaux auxiliaires sont actuellement les attaques les plus efficaces contre ces systèmes. Contrairement aux attaques classiques qui n'exploitent que les entrées et sorties des algorithmes, elles utilisent également les fuites physiques du composant. Elles se servent par exemple de la température, du rayonnement, ou de la consommation de courant. La recherche dans cette discipline est donc un enjeu capital de sécurité.

Les attaques par canaux auxliaires ont été introduites par Kocher en 1996 [8]. Il a été le premier à établir un lien entre le temps de calcul d'un algorithme et ses données secrètes. Dans les années qui ont suivi, plusieurs principes d'attaques utilisant des statistiques sur la consommation de courant ont émergé ([9], [4] et [10]). Celles-ci ont rapidement été implémentées de manière efficace. Face aux résultats significatifs de ces attaques, le monde industriel s'est impliqué dans cette recherche à partir des années 2000. Son but a été de protéger les systèmes embarqués dont les applications sont civiles (documents d'identité, cartes bancaires..) ou militaires (communication entre gouvernements, transmissions militaires...). Depuis les années 2000, les systèmes cryptographiques ne cessent d'évoluer pour acquérir de nouvelles contre-mesures (voir [6] et [5] par exemple) dont le rapport efficacité/sécurité varie en fonction de l'usage du composant. La thèse de Sonia Belaïd [2], une de mes encadrantes chez Thales, a été centrée sur les idées de contre-mesures de ces attaques d'un point de vue théorique. Pour faire avancer rapidement la recherche, plusieurs entités académiques ou industrielles ont lancé des concours publics d'attaques avec des schémas de plus en plus sécurisés. Un des plus connu est le **DPA Contest** [1] lancé par Télécom Paristech en 2008. Quatre concours se sont succédés avec des schémas de plus en plus difficiles à attaquer. Le **CHES Challenge** [2] a aussi été lancé pendant mon stage. Il s'agit d'une plateforme où les participants peuvent soumettre ou attaquer des schémas de manière ludique. Ce challenge a été lancé avant la conférence CHES 2016 qui s'est déroulée en août à Santa Barbara.

Le problème étudié

L'objectif de mon stage a été d'étudier la sécurité contre les attaques par consommation de puissance du schéma de chiffrement symétrique AES. Une particularité intéressante de ce projet réside dans le fait qu'il se voulait très concret : les nouvelles idées d'attaques devaient être expérimentalement validées. C'est-à-dire qu'elles devaient être physiquement implémentées et fonctionnner sur des données réelles. Et, les nouvelles contre-mesures devaient être étudiées en gardant le compromis efficacité/sécurité à l'esprit. La question de la sécurité de l'AES est cruciale puisque cet algorithme est utilisé pour chiffrer la plupart des données sur composants électroniques à destination civile. Sa sécurité a été très étudiée ces dernières années et les concours d'attaques par canaux auxiliaires ciblent en grande majorité l'AES. Ce domaine de recherche étant très dynamique et nouveau, j'étais motivée pour essayer d'y contribuer à mon niveau. Le stage s'est composé d'une succession de phases de **recherche d'attaques** et de phases de **recherche de contre-mesures**. Après avoir attaqué de manière classique l'AES non sécurisé, j'ai étudié les contre-mesures telles que la randomisation et le masquage. J'ai ensuite intégré ces contres-mesures dans l'AES et ai procédé à de nouvelles attaques pour en vérifier la résistance. Il s'est avéré que de nouvelles contres-mesures ont été nécessaires. J'ai donc recherché des nouvelles idées d'améliorations qui ne nuisent pas l'efficacité. Une phase d'évaluation de ces nouveaux schémas s'en est suivi puis une proposition publique.

La contribution proposée

Les résultats importants de mon stage sont les suivants :

— la **création d'une attaque sur le dernier schéma d'AES proposé par le DPA Contest**. Ce schéma a été conçu pour être protégé contre un certain type d'attaques. Nous avons réussi à contourner cette

1. http://www.dpacontest.org/home/index.html
2. https://ctf.newae.com/

protection et l'attaque a été classée parmi les meilleures dans sa catégorie. Nous avons rendu possible l'attaque théoriquement empêchée grâce à des ajouts d'attaques auxiliaires.

— La proposition d'une **amélioration du dernier schéma du DPA Contest comportant des nouvelles contre-mesures**. Nous avons étudié les failles du schéma précédemment attaqué et avons proposé des contre-mesures qui préservent l'efficacité. Les idées ont été inspirées de papiers théoriques et ont été adaptées pour être réalisables techniquement. A notre connaissance, ces propositions sont plus difficiles à attaquer. Il est important de rappeler que cette solution ne prétend pas non plus contrer toutes les attaques publiées. En effet, les codes des autres attaques ne sont pas publics et seule une description succinte est disponible. Cependant, une grande partie des attaquants ont expliqué avoir eux-aussi tiré parti des failles trouvées. Les contre-mesures proposées sont donc un compromis sécurité/efficacité qui semble pouvoir empêcher plusieurs attaques.

Les arguments en faveur de sa validité

Tout d'abord, pour valider l'attaque du DPA Contest, nous avons fait des tests personnels sur une base de données publique. Elle a ensuite été évaluée par l'équipe de Télécom Paristech sur la base de données secrète et ses performances ont été publiées. Par ailleurs, la validation du schéma comportant les nouvelles contre-mesures a été plus longue. La solution a été évaluée de différentes manières. J'ai en premier lieu utilisé un outil de mesure permettant d'attaquer ce nouveau schéma. Il fallait prendre certaines précautions pour pouvoir comparer les résultats obtenus avec les autres schémas. Cette phase a permis d'effectuer plusieurs optimisations et d'invalider certaines hypothèses. Les résultats finaux étant satisfaisants, le schéma a été soumis sur la plateforme du CHES Challenge. La soumission des schémas et les attaques publiquées m'ont permis de remporter le prix *Best Student CHES Challenge 2016*. Ceci a entrainé des échanges avec des experts en sécurité. Le schéma ayant bien résisté sur la plateforme, un travail de mise en commun du code a été amorcé avec les organisateurs DPA Contest.

Le bilan et les perspectives

L'étape suivante sera une seconde mise à l'épreuve des contre-mesures, l'idée étant d'en faire un nouveau concours d'attaque du DPA Contest. Ce schéma comporte un rapport sécurité/efficacité adapté pour des applications civiles. Dans mon dernier mois de stage, je vais aussi étudier la sécurité d'un schéma de chiffrement utilisant une contre-mesure forte inspirée de papiers théoriques publiés à CHES 2016. La sécurité sera théoriquement beaucoup plus forte et applicable dans le domaine militaire. Cela entrainera des coûts en temps de calcul que l'on ne cherchera pas pour l'instant à minimiser. Une perspective de recherche pourrait être l'étude et la modification de ces contre-mesures pour les rendre efficaces et adaptées à des implémentations hardware. Ce stage m'a apporté une formation complémentaire à celle des cours de cryptologie au MPRI et à Télécom Paristech. Maintenant, je suis capable de passer rapidement à la pratique pour valider ou invalider les pistes de recherche. J'ai aussi associé la dimension pratique (efficacité, temps de calcul, différence hardware/software) aux algorithmes théoriques étudiés. J'ai pratiqué beaucoup de langages de programmation (C, C++, Unix, Assembleur, Python). Je crois être maintenant capable de choisir le langage adapté à la preuve expérimentale voulue. J'ai pris plaisir à chercher des nouvelles attaques et contre-mesures. Avec l'entrainement dû aux concours d'attaques, je crois avoir développé un "détecteur de fuites d'information" qui me permet de repérer les points possibles d'attaques horizontales ou verticales. Finalement, les canaux auxiliaires sont la plus grande menace pour les systèmes cryptographiques actuels. Je ne l'oublierai pas si je suis amenée à concevoir des systèmes cryptographiques.

Ce stage a aussi été riche en rencontres professionnelles. J'ai échangé avec beaucoup d'acteurs dans le domaine : chercheurs de Thales, professeurs à Télécom Paristech, chercheurs à l'ENS, experts en cryptologie à l'ANSSI, experts en sécurité à Ingenico et Gemalto et même au Canada ! Ceci m'a permis de mûrir un projet de thèse Cifre avec Thales, l'ENS Paris et l'ANSSI. Je commencerai ma thèse au printemps 2017 après un projet de recherche de 6 mois chez Cryptography Research à San Francisco.

Table des matières

1 Motivations et définitions

A partir des années 2000, la sécurité contre les attaques physiques est devenue une réelle contrainte pour les concepteurs de systèmes embarqués. Ces attaques "à part" ne remettent pas en cause la sécurité mathématique mais elles utilisent des données physiques (temps de calcul, température, courant consommé...) pour retrouver la clé secrète. Elles deviennent actuellement de plus en plus sophistiquées et nécessitent de moins en moins de mesures. Des cartes bleues aux puces de transmissions militaires en passant par les documents d'identités, ces attaques menacent maintenant tous les systèmes cryptographiques embarqués. Ce domaine de recherche est donc en plein essor.

On distingue les attaques actives des attaques passives. Les premières consistent à modifier le système attaqué. Ces attaques sont en général irréversibles et utilisent des découpages chimiques ou des lasers. Les attaques passives utilisent des mesures de paramètres physiques (temps, température, consommation de puissance, rayonnement) appelés canaux auxiliaires, pour en déduire des informations sur les données secrètes.

L'objet de ce projet s'est porté sur une des attaques les plus étudiées et les plus menaçantes actuellement. Il s'agit d'aborder la **sécurité du système de chiffrement AES contre les attaques par canaux auxiliaires utilisant la consommation de puissance**. Ce qui m'a fascinée dans cette étude a été la possibilité de retrouver des informations extrêmement sécurisées mathématiquement en utilisant de la physique. J'ai ainsi compris qu'avant d'être implémenté électroniquement, tout schéma théoriquement sûr devait aussi être analysé du point de vue des fuites physiques. Cette partie introduit les données nécessaires à l'attaquant ainsi que trois attaques par consommation de puissance.

1.1 Traces de consommation

Pour mener une attaque par canaux auxiliaires utilisant la consommation de puissance, un attaquant doit réaliser des mesures de l'équipement lorsque l'algorithme visé est exécuté.

Définition 1. *On appelle trace, un graphe de mesure de consommation de courant ou de rayonnement d'un appareil exécutant un algorithme en fonction du temps.*

La plupart des attaques voient leur taux de succès augmenter avec le nombre de traces. Pour mesurer les traces de consommation, une technique est de mesurer le voltage autour d'une résistance à l'entrée du CPU [3]. Certains outils avec composant programmable permettent de mesurer ceci. J'avais à disposition un tel système de mesure que j'ai utilisé en deuxième partie de stage.

Pour la suite, il convient de différencier deux types d'implémentations embarquées. Tout d'abord, l'implémentation dite *hardware* est une implémentation bas niveau où le concepteur crée lui-même sa puce. Une implémentation *software* est l'implémentation d'un algorithme sur un processeur. Dans une implémentation AES *software*, la taille maximale d'un registre étant restreinte, les octets de l'entrée sont souvent traités les uns après les autres de manière séquentielle. La trace obtenue est la consommation du traitement de chaque octet les uns après les autres. Dans une implémentation AES *hardware*, les contraintes sont différentes et tous les octets (blocs de 8 bits) de l'entrée peuvent être traités en parallèle en utilisant des plus gros registres. C'est le cas des systèmes que j'ai étudiés. La trace obtenue est la consommation du traitement des 16 octets en simultanés. Les traces sont donc plus courtes qu'en *software* mais contiennent moins d'informations.

1.2 Les Attaques SPA

Les attaques par consommation de puissance les plus simples sont appelées SPA (*Single Power Analysis*). Elles exploitent l'information observée sur une ou très peu de traces de consommation. La consommation est généralement reliée aux changements de l'état de registres donc à la manipulation de données et aux processus de calculs. Ainsi, on peut reconnaître la trace de consommation d'un AES-128 implémenté en hardware puisqu'il réalise dix fois les mêmes calculs et on reconnait donc 10 patterns similaires.

3. Dans tout ce rapport, la consommation est mesurée de cette manière.

En 1996, Kocher [8] a montré que si les calculs dépendent des valeurs des bits de la clé secrète, la trace de consommation donne à priori directement la clé. Le schéma d'une telle trace est donné en figure 1 [2]. Les grands (resp petits) pics représentent la consommation de puissance générée durant les opérations qui utilisent un bit de clé égal à un (resp zéro). En identifiant les tailles des pics, on peut retrouver entièrement la clé. L'inconvénient de ces attaques est qu'**elles sont peu resistantes au bruit**.

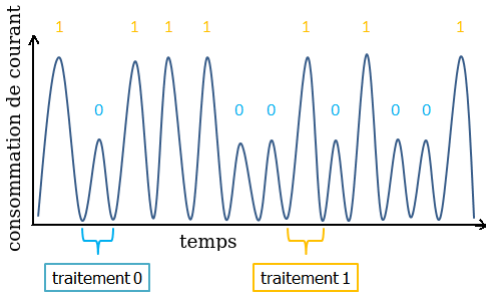

FIGURE 1 – Attaque SPA sur un algorithme dont la consommation dépend des valeurs de la clé (1011100101001)

1.3 Les Attaques DPA

En 1999, Kocher, Jaffe et Jun ont publié une nouvelle attaque [9] beaucoup plus puissante qu'une attaque SPA, la DPA (Differential Power Analysis). L'idée est de combiner l'information collectée sur plusieurs traces correspondant à plusieurs entrées. Voici le descriptif des étapes de cette attaque (expliquées dans le livre [11]). Celui-ci est aussi schématisé en figure 2.

0. Phase préliminaire : Il est nécessaire de **mesurer plusieurs traces** de consommation de l'algorithme utilisant la clé inconnue et des entrées connues. On suppose qu'il y en a N. Par exemple, dans le cas d'une attaque pendant un chiffrement AES, on dispose pour chaque trace $i \in [1, N]$ de la donnée du clair M_i et/ou du chiffré C_i.
1. Phase de prédiction : Il faut établir un **modèle de prédiction**, aussi appelé modèle de fuite. Cela consiste à :
   (a) Choisir une étape de l'algorithme à attaquer et trouver l'instant t correspondant dans les traces, appelé *instant d'interêt*.
   (b) Etablir une dépendance entre la consommation de puissance $W(t)$ à cet instant et la valeur intermédiaire manipulée à cette étape de l'algorithme.

   C'est l'étape la plus difficile dans l'élaboration d'une attaque DPA. En effet, pour que l'attaque réussisse, la valeur intermédiaire manipulée à l'instant ciblé doit dépendre :
   — d'une petite partie de la clé (un octet par exemple) puisqu'il faudra ensuite lister toutes les valeurs possibles.
   — d'une entrée. En effet, sinon le fait d'avoir plusieurs traces n'apporte pas d'information.

   On note $V_{i,K}$ la valeur intermédiaire choisie dépendant de l'entrée $i \in [0, N]$ et de la clé K. On notera $\tilde{K}$ lorsqu'il s'agit d'une hypothèse de clé, et K lorsqu'il s'agit de la clé secrète.

   Par exemple, la valeur intermédiaire peut être l'état en sortie de la première boîte S : $S[M \oplus K]$. Et on peut lui associer une dépendance binaire entre son bit de poids faible et la consommation de puissance à l'instant correspondant. En supposant que l'on attaque le premier octet de clé K, le premier octet du message i étant écrit M_i, une dépendance binaire peut s'écrire :

$$\forall i \in [0, N] \text{ (pour chaque trace) } W_i(t) \begin{cases} \geq 0 \text{ si } S[M_i \oplus K] = 0 \bmod 2 \\ \leq 0 \text{ si } S[M_i \oplus K] = 1 \bmod 2 \end{cases}$$

   Ensuite, il faut établir des **prédictions** : Pour chaque valeur possible de la petite partie de clé attaquée $\tilde{K}$, on calcule les N valeurs intermédiaires de l'algorithme. C'est-à-dire que l'on calcule $V_{i,\tilde{K}}$ pour tous

$i \in [1, N]$ et pour tous les $\tilde{K}$.
Dans l'exemple du bit de poids faible, pour chaque hypothèse sur $\tilde{K} \in [0, 255]$, cela revient à trier les traces en deux groupes $E_{\tilde{K}}[1]$ et $E_{\tilde{K}}[0]$, celles dont la prévision est 1 et celles dont la prévision est 0.

$$E_{\tilde{K}}[0] = \{i \mid S[M_i \oplus \tilde{K}] = 0 \bmod 2\} \text{ et } E_{\tilde{K}}[1] = \{i \mid S[M_i \oplus \tilde{K}] = 1 \bmod 2\}.$$

2. Phase de distinction : Pour déterminer la clé secrète, on utilise un **distingueur**. C'est une méthode de classement des hypothèses de clés en fonction des consommations mesurées et du modèle de prédiction. En suivant l'exemple, on peut distinguer les clés en calculant la différence des moyennes des deux groupes.

$$Distingueur(\tilde{K}) = \frac{1}{|E_{\tilde{K}}[0]|} \sum_{i \in E_{\tilde{K}}[0]}^{n} W_i(t) - \frac{1}{|E_{\tilde{K}}[1]|} \sum_{i \in E_{\tilde{K}}[1]}^{n} W_i(t)$$

Si cette différence s'éloigne de 0, cela veut dire que les groupes ont une moyenne de consommation différente et que la séparation a été pertinente. L'hypothèse de clé est d'autant plus valable que cette différence est importante.

$$K = \underset{\tilde{K} \in [0,255]}{\operatorname{argmax}} |Distingueur(\tilde{K})|$$

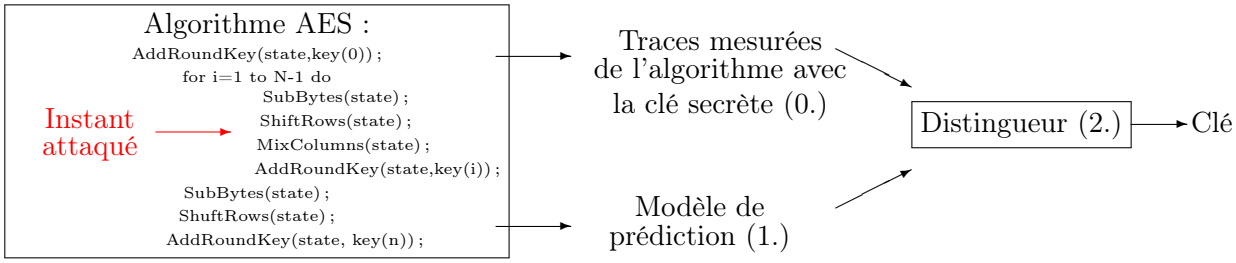

FIGURE 2 – Schéma du principe de l'attaque DPA

1.4 Les Attaques CPA

L'attaque CPA (Corrélation Power Analysis) a été introduite plus tard (voir [4] et [10]). C'est **une attaque DPA avec un distingueur particulier : le coefficient de corrélation de Pearson**. Voici les caractéristiques d'une attaque CPA :

1. Soit t l'instant correspondant à l'étape choisie comme cible. L'attaque CPA suppose une **dépendance linéaire** :

$$W_i(t) = \lambda \cdot f(V_{i,K}) + \beta \qquad \text{(Modèle de fuite CPA)}$$

$W_i(t)$ est la consommation en puissance à l'instant t pour la trace i, λ et β sont des coefficients inconnus indépendants de i, $f(V_{i,K})$ est une fonction de la valeur intermédiaire $V_{i,K}$ manipulée à l'étape ciblée qui dépend de la trace i et de la clé K. La fonction f peut par exemple être l'identité, le poids de Hamming ou une distance de Hamming.
Ensuite, on calcule $f(V_{i,\tilde{K}})$ pour toutes les hypothèses de clé $\tilde{K}$ (en ciblant un octet, il y a 256 possibilités) et pour les N mots correspondant aux traces disponibles :

$$\begin{pmatrix} f(V_{0,1}) & f(V_{1,1}) & \cdots & f(V_{255,1}) \\ f(V_{0,2}) & f(V_{1,2}) & \cdots & f(V_{255,2}) \\ \vdots & \vdots & & \vdots \\ f(V_{0,N}) & f(V_{1,N}) & \cdots & f(V_{255,N}) \end{pmatrix} \qquad \text{(Prédictions)}$$

2. Enfin, pour trouver la colonne de la matrice qui correspond le mieux au modèle de fuite et ainsi retrouver la clé, on utilise le **coefficient de corrélation de Pearson**. C'est le distingueur propre aux attaques CPA.

$$\rho(\tilde{K}, t) = \frac{Cov(f(V_{i,\tilde{K}})_{1\leq i\leq N}, W_i(t)_{1\leq i\leq N})}{\sigma_{(f(V_{i,\tilde{K}}))_{1\leq i\leq N}} \cdot \sigma_{(W_i(t))_{1\leq i\leq N}}} \qquad \text{(Coefficient de Pearson)}$$

Avec Cov correspondant à la covariance. Plus le coefficient de Pearson est proche de 1 ou -1, plus il existe de dépendance linéaire entre $f(V_{i,\tilde{K}})_{1\leq i\leq N}$ et $W_i(t)_{1\leq i\leq N}$.
On choisit donc la clé qui maximise ce coefficient de Pearson.

$$K = \underset{\tilde{K}}{\text{argmax}} |\rho(\tilde{K}, t)|$$

En résumé, le coefficient de corrélation est un moyen efficace de retrouver la clé quand la dépendance est bien linaire. La difficulté reste de trouver l'étape où la valeur intermédiaire dépend d'une petite partie de la clé et du clair ou du chiffré et de définir la fonction f. Les attaques appliquées dans la suite sont des combinaisons d'attaques SPA et CPA adaptées à chaque implémentation de l'AES.

2 Attaque sur un AES non-sécurisé

Durant cette première phase d'attaque, j'ai étudié un AES ne comprenant aucune contre-mesure contre les attaques par canaux auxiliaires. Ce type d'AES était implémenté dans les systèmes embarqués dans les années 2010 avant d'y intégrer des contre-mesures. J'ai tout d'abord récupéré des traces de consommation grâce à la plateforme en ligne du DPA Contest. Ensuite, il a fallu choisir le type d'attaque et le modèle de prédiction. Ces choix ont été orientés par le travail des précédents stagiaires.

2.1 Traces de consommations du DPA Contest V2

Pour mener une attaque sur un AES non sécurisé, j'ai utilisé les données du DPA contest V2 dont un exemple de trace est en figure 3.

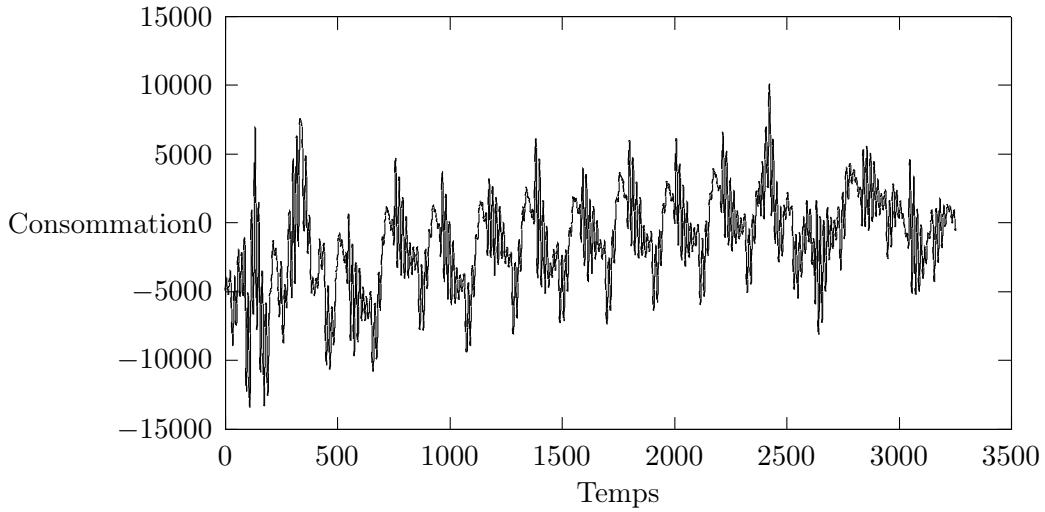

FIGURE 3 – Une trace de consommation d'un chiffrement AES non protégé extraite du DPA Contest V2

Le DPA contest V2 est une banque de données de consommation d'un AES-128 mise en ligne par Télécom Paristech. Cette banque fait partie du concours DPA Contest dont le but est de mener des attaques par canaux auxiliaires et ainsi comparer leur efficacité de manière objective. En effet, lorsque chaque laboratoire mesure ses propres traces, il est difficile de comparer les attaques menées car trop de paramètres sont variables : la sensibilité de la plate-forme d'acquisition, l'implémentation de l'algorithme, le bruit. Le concours pour cette version non sécurisée s'est terminé en 2011.

L'implémentation du DPA Contest V2 est hardware. On reconnaît sur la trace en figure 3, 10 patterns qui se répètent (entre $t = 500$ et $t = 2500$) qui correspondent aux 10 tours de l'AES. Sans connaître l'implémentation hardware, il est difficile de dire quelle étape d'un tour consomme le plus. Mais d'une manière générale, c'est le stockage dans les registres qui consomme le plus. Il a été constaté empiriquement que la consommation était corrélée au poids de Hamming des valeurs qui sont stockées. Plus précisement, elle est liée à la distance Hamming entre la nouvelle valeur et l'ancienne valeur du registre. Lorsqu'on a accès à l'ancienne valeur, on procède à une attaque dite *en distance de Hamming*.

L'attaque menée est une attaque CPA classique en distance de Hamming au niveau du dernier tour.

2.2 Modèle de prédictions

Suivant la définition de l'attaque CPA en partie 1.4, il convient de choisir une valeur intermédiaire vulnérable et d'établir une fonction f.

Pour la fonction f, on a choisi une distance de Hamming entre la valeur intermédiaire et la valeur qui la remplace dans son registre. Nous avons choisi de ne pas étudier l'implémentation bas niveau, et de chercher à l'aveugle quelle valeur remplace quelle autre dans les registres. J'ai testé l'attaque sur plusieurs distances au niveau du dernier tour. Le modèle le plus concluant était celui-ci défini en figure 4. La valeur intermédiaire vulnérable est l'entrée de la dernière Sbox :

$$V_{i,\tilde{K}} = S^{-1}(C_i \oplus \tilde{K}) \quad \text{(Valeur intermédiaire choisie)}$$

Tous les octets sont traités en même temps dans 16 circuits différents. Pour la même consommation à l'instant d'interêt, on considère les 16 modèles de prédictions séparément. C'est-à-dire que l'on intègre les 15 octets non considérés dans les constantes λ et β. Le modèle de prédiction est le suivant :

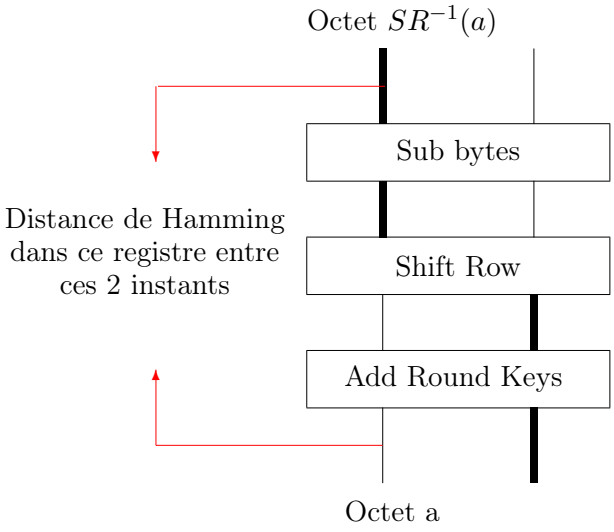

FIGURE 4 – Dernier tour de l'AES

$$\forall a \in [0, 15] : f(V_{i,\tilde{K_a}}) = HW(S^{-1}(C_{i,a} \oplus \tilde{K_a}) \oplus C_{i,SR^{-1}(a)}) \quad \text{(Modèle de prédiction)}$$

où HW est la fonction qui associe le poids de Hamming, S est l'opérateur Subbytes, $C_{i,a}$ est l'octet a du chiffré i, SR est l'opérateur ShiftRow.

Instant d'interêt

Comme toutes les courbes sont synchronisées, on cherche l'instant précis où le registre passe de $S^{-1}(C_{i,a} \oplus K_a)$ à $C_{i,SR^{-1}(a)}$. C'est par définition l'instant où la comsommation de puissance suit le mieux le modèle de fuite. La connaissance de cet instant permet de réduire l'étude des puissances consommées en un seul point. Pour le trouver, il s'agit de faire une **phase d'apprentissage** qui consiste à calculer $\rho(K, t)$ (avec K la bonne

hypothèse de clé) à tous les instants comme en figure 5. L'instant d'interêt est l'instant où la corrélation est la plus forte.

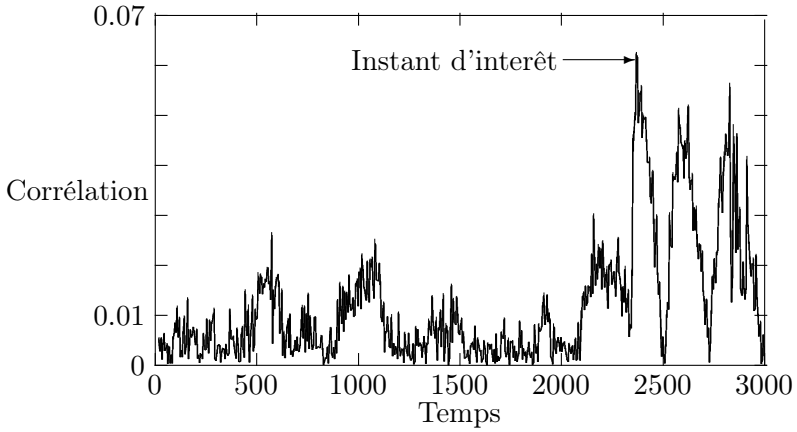

FIGURE 5 – Recherche de l'instant d'interêt

En pratique, il est difficile d'avoir un jeu de traces où l'on connait la clé pour y faire un apprentissage. Ne connaissant pas l'instant d'interêt, l'attaque doit s'effectuer à tous les instants du dernier tour. On obtient une hypothèse de clé pour chaque instant et on renvoie la clé qui revient le plus souvent. Cette attaque s'appelle attaque aveugle.

2.3 Attaque

Après une mise à niveau en C, j'ai crée un code permettant de lire les courbes et ai implémenté l'attaque. Le pseudo-code de l'attaque CPA que j'ai implémentée est placé en annexe A. Les résultats que j'ai obtenus sont rassemblés dans la table 1.

TABLE 1 – Résultats obtenus lors de l'attaque du DPA Contest V2

Type d'attaque	Nombre de courbes	Octets de clé retrouvés
Avec apprentissage	20000	98%
Aveugle	20000	90%
Aveugle	10000	84%
Aveugle	5000	59%

2.4 Attaque améliorée

L'AES étant implémenté en *hardware*, chaque boîte S correspond à un circuit différent fonctionnant en même temps. Le modèle peut donc être amélioré en considérant les fuites des différentes boîtes S. C'est-à-dire qu'on prend en compte les fuites correspondant aux octets $b \in [0, a-1]$ précédent a. Les octets de clés trouvés précédement sont notés K'_b.

$$f(V_{i,\tilde{K_a}}) = \sum_{b=0}^{a-1} HW(S^{-1}(C_{i,b} \oplus K'_b) \oplus C_{i,SR^{-1}(b)}) + HW(S^{-1}(C_{i,b} \oplus \tilde{K}_a) \oplus C_{i,SR^{-1}(a)})$$

(Modèle de fuite amélioré)

Ainsi pour l'octet 0, l'attaque est inchangée, mais plus on avance dans les octets, plus le modèle est précis, à condition que les octets de clés trouvés soient en majorité corrects. Les résultats que j'ai obtenus sont rassemblés dans la table 2.

TABLE 2 – Résultats obtenus lors de l'attaque améliorée du DPA Contest V2

Nombre de courbes	Sans amélioration	Avec amélioration
20000	90%	92%
10000	84%	92%
5000	59%	65%

C'est ce principe d'attaque CPA améliorée qui a permis à Matthieu Walle (ancien stagiaire du même laboratoire) de remporter le DPA Contest V2 en 2011 avec 7000 traces pour obtenir 80% de la clé. Le site est toujours ouvert pour soumettre des attaques et depuis, d'autres attaques ont permis d'établir le record à 439 courbes pour obtenir 80% de la clé. Ces nouvelles attaques utilisent la méthode des templates qui sera expliquée par la suite.

3 Le masquage et la randomisation comme contre-mesures

3.1 La randomisation

Les premières contre-mesures proposées ont été de randomiser les calculs. Si chaque octet est traité à un instant aléatoire, l'instant à prendre diffère à chaque trace. La randomisation est une contre-mesure localisée, c'est-à-dire qu'elle cible une étape vulnérable précise. Voici les deux idées de randomisation principales :

— Le *Jitter* est une contre-mesure qui consiste à faire attendre l'aglorithme pendant un temps aléatoire avant chaque calcul sensible. Les traces sont alors désynchronisées. Pour contourner cette contre-mesure, il faut opérer à un resynchronisation des traces.

— Le *Shuffle* est une contre-mesure qui randomise l'ordre de traitement des octets pour une opération sensible donnée. Cette contre-mesure est possible lorsque les octets sont traités d'une manière software les uns après les autres. Elle est plus efficace que le *jitter* puisque l'ordre étant inconnu, l'attaquant est incapable d'associer les instants de traitement aux bons octets.

3.2 Le masquage

Le masquage est la contre-mesure la plus répandue. Il a été introduit en 1999 simultanément par Goubin et Patarin [6], et Chari [5]. L'idée du masquage est d'**éviter que, tout au long de l'algorithme, les valeurs intermédiaires ne dépendent directement des valeurs sensibles** (sortie de boites S par exemple). Pour que l'attaquant n'y ait pas accès, celles-ci vont être dédoublées en plusieurs variables paraissant aléatoires appelées masques.

Définition 2. Masquage booléen d'ordre d *: Un masquage booléen d'ordre d est l'opération qui remplace une donnée x appelée donnée sensible par $d+1$ données y_0 ,..., y_d où*

$$\begin{cases} y_0 \leftarrow \$ \\ \vdots \\ y_{d-1} \leftarrow \$ \\ y_d = x \oplus y_0 \oplus y_1 \oplus ... \oplus y_{d-1} \end{cases} \tag{1}$$

Les y_i, sont appelés les masques

Définition 3. Attaque CPA d'ordre d *: Une attaque CPA d'ordre d est une attaque CPA qui s'applique sur une combinaison de d mesures de consommation sur la trace.*

Dans le cas de l'ordre 2, la combinaison utilisée est souvent la différence en valeur absolue. D'une manière générale, plusieurs fonctions de combinaisons existent et utilisent la multiplication, la soustraction ou la mise au carré.

Généralement, **on considère qu'un schéma est sûr lorsque l'attaquant n'obtient aucune information en observant n'importe quel ensemble d'au plus d variables intermédiaires**. En effet, chaque valeur de masque induit une distribution qui ne dépend statistiquement pas de la valeur intermédiaire non masquée. Par exemple, la distribution $x \oplus y_0$ est toujours la même, quelque soit la valeur de x. La seule manière d'avoir accès à la valeur sensible x est de retrouver $y_0 \oplus y_1 \oplus ... \oplus y_d$.

Par ailleurs, plus l'ordre de l'attaque est élevé, plus il y a de masques et moins il est facile de retrouver la clé. Pour des applications civiles, le masquage utilisé est à priori d'ordre petit (1 ou 2). Tandis que pour des applications militaires, les masquages sont à priori d'ordre supérieurs à 2 pour augmenter le niveau de sécurité au dépens de l'efficacité.

Construire un masquage d'ordre 1 pour l'AES

Pour masquer un AES, il convient de modifier ses blocs. En effet, la valeur sensible d'entrée x subit tout de suite une étape de masquage qui la dédouble en $x \oplus R$ (le masqué) et R (le masque).

Tout au long de l'algorithme, au lieu de manipuler une donnée, **l'AES doit manipuler deux données parallèles : le masqué et le masque**. Un tel bloc est schématisé en figure 6. Les blocs de l'AES doivent être modifiés de manière à avoir deux entrées et deux sorties et suivant des contraintes :

- entrée$_1 \oplus$ entrée$_2$ = valeur sensible d'entrée
- sortie$_1 \oplus$ sortie$_2$ = valeur sensible de sortie
- ne jamais manipuler entrée$_1 \oplus$ entrée$_2$ ni sortie$_1 \oplus$ sortie$_2$ qui correspondent aux données sensibles.

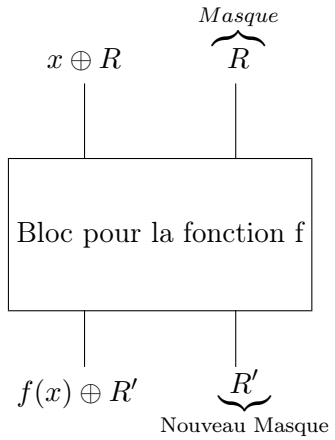

FIGURE 6 – Bloc pour un masquage d'ordre 2

Lorsqu'un bloc f est linéaire pour le $\oplus$ (*ShiftRows* ou *MixColumns* par exemple), il suffit de l'appliquer au masqué et au masque simultanément. En effet $f(x \oplus R) \oplus f(R) = f(x) \oplus f(R) \oplus f(R) = f(x)$.

Pour les opérations non linéaires (boîte S par exemple), il faut les concevoir en gardant les contraintes à l'esprit. Plusieurs idées d'implémentations plus ou moins coûteuses ont été introduites. Pour la boîte S, une solution simple mais coûteuse en mémoire consiste à pré-calculer les 256 S-box $S_i^*(\cdot) = S(\cdot \oplus i)$ appelées les

"look-up tables". Il reste à définir un nouveau masque R' et à prendre $S^*_{x\oplus R}(R) \oplus R'$.

Lorsque l'on veut masquer l'AES avec un masquage booléen, à chaque fois que l'on définit un nouveau masque, on tire une valeur aléatoire de 8 bits. Ce type de masquage oblige l'attaquant à faire une attaque d'ordre 2 mais est aussi coûteux.

Le *coût* d'un algorithme peut se mesurer en taille de code, en nombre de cycles de processeurs ou encore en nombre d'aléas générés. La minimisation de tous ces paramètres est importante pour rendre l'implémentation potentiellement applicable dans le domaine civil. La table 3 répertorie les résultats trouvés (en nombre de cycles) par Rivain et Prouff [13] pour leur implémentation de masquage sur une même plateforme.

TABLE 3 – Comparaison d'efficacité pour les masquages de Rivain et Prouff

Algorithme	Perte de vitesse par rapport à un AES non protégé
Masquage booléen d'ordre 1	x43
Masquage booléen d'ordre 2	x90
Masquage booléen d'ordre 3	x156

Le RSM : un compromis de masquage pour l'AES

Le *Rotating Sboxes Masking* a été introduit en 2012 par Nassar *et al* [14]. Cette technique utilise **un ensemble fixe choisi de 16 masques de 8 bits** $(m_i)_{0\leq i\leq 15}$.
L'aléatoire vient d'un entier appelé **offset** qui est l'indice du masque utilisé. Celui-ci est tiré aléatoirement en début de chiffrement. A chaque fois que l'algorithme a besoin de générer un nouveau masque, l'*offset* est ensuite incrémenté modulo 16.

A la manière d'un barillet (figure 7), l'*offset* pointe le masque courant. Beaucoup moins de bits aléatoires sont générés et la distribution des probabilités des valeurs sensibles pour un masqué donné reste étudiée pour qu'il soit difficile de retrouver la valeur sensible.

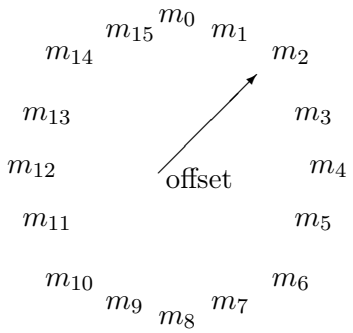

FIGURE 7 – Barillet

3.3 Le DPA Contest V4.2

Suite aux résultats du DPA Contest V2, l'équipe de Télécom Paristech a crée un nouveau schéma d'AES à base de masquage RSM (voir [3]). Un concours d'attaque sur ce schéma a été lancé en 2015 : le DPA Contest V4.2 [4] . Cette fois, l'implémentation est un AES software utilisant le RSM avec les masques tournants : 0x03, 0x0c, 0x35, 0x3a, 0x50, 0x5f, 0x66, 0x69, 0x96, 0x99, 0xa0, 0xaf, 0xc5, 0xca, 0xf3, 0xfc.
En voici une brève description.

Plusieurs *offsets*

Pour éviter qu'un unique *offset* ne soit facile à retrouver, l'implémentation est telle que chaque octet de l'état possède sont propre offset. Il y a alors 16 barillets, et $4 \cdot 16 = 64$ bits aléatoires générés en début de

4. Les versions entre V4 et V4.2 étaient aussi des implémentations de l'AES-RSM mais comportaient d'autres failles

chiffrement. Pour l'étape des boîtes S, l'AES utilise 16 boîtes S pré-calculées S^* qui sont conçues pour vérifier :

$$\forall (i,j) \in [0,15]^2 \ S^*(x \oplus m_j, m_j) = (S_{AES}(x) \oplus m_{j+1}, m_{j+1})$$

Shuffling

Le DPA Contest V4.2 étant traité en *software*, les opérations sur les octets sont faites séquentiellement. Pour éviter d'attaquer au premier et au dernier tour, un *Shuffle* définit leur ordre de traitement. Cette permutation aléatoire est tirée en début du chiffrement.

Le pseudo-code de l'algorithme est disponible en annexe B. Par rapport à un AES non protégé, j'ai trouvé que l'AES du DPA Contest V4.2 est 4 fois moins rapide. Toutefois, il est important de noter que cette valeur a été trouvée en comparant les deux implémentations sur ma plateforme. La comparaison de cette perte de vitesse avec celles de la table 3 est risquée puisque les implémentations ne sont pas optimisées pour le même équipement.

4 Attaque de l'AES-RSM du DPA Contest V4.2

Le DPA Contest V4.2 a été étudié pour contrer les attaques proposées pour les versions antérieures. La recherche d'attaques sur cette implémentation est toujours active et la plateforme du DPA Contest est ouverte aux soumissions. Une fois une attaque soumise, ses performances sont rendues publiques mais seulement son titre est publié. La plupart des attaques semblent d'être d'ordre 2 et utilisent des méthodes de *machine learning* comme les SVM.

L'idée a été de faire une attaque d'ordre 1 en utilisant des attaques auxiliaires très optimisées. La partie de l'attaque utilisant une CPA est dite *verticale* puisqu'elle combine les informations de plusieurs traces. Les attaques auxiliaires utilisent de la SPA sont appelées *horizontales* puisqu'elles utilisent seulement les informations contenues dans la trace en question. L'attaque obtenue a été publiée sur le site du DPA Contest. Elle a été placée dans les meilleures attaques en terme de nombre de traces et de temps. Ce qui a été excitant dans cette partie du stage a été de réussir à montrer qu'une attaque d'ordre 1, lorsqu'elle est très optimisée, peut être meilleure que d'autres types d'attaques d'ordre plus élevés.

4.1 Idée de l'attaque

J'ai mené une attaque CPA d'ordre 1 en poids de Hamming au premier tour. L'attaque en distance de Hamming est impossible à cause de la remise à zéro des registres entre les étapes. Nous procédons à une attaque en poids de Hamming à la sortie de la première boîte S comme schématisé en figure 8.

La valeur intermédiaire attaquée est :

$$V_{i,\tilde{K_a}} = S[Clair_a \oplus \tilde{K_a}] \oplus M_{offset_a+1}$$

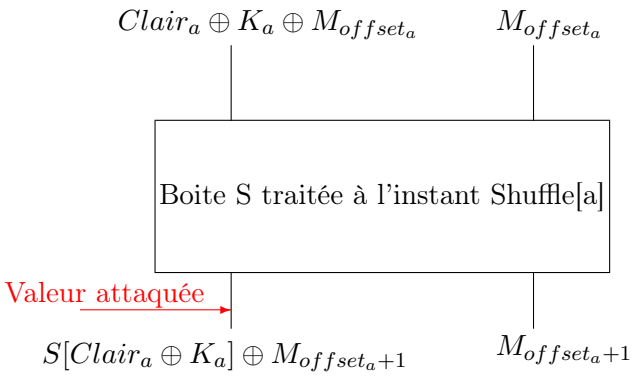

FIGURE 8 – Sortie de la boîte S au premier tour

Tous les octets sont traités les uns après les autres. Pour chaque octet a au sein d'une même étape, l'instant d'interêt t_a est différent et se trouve avec une phase d'apprentissage. Le modèle de prédiction est le suivant :

$$\forall a \in [0,15] : f(V_{i,\tilde{K_a}}) = HW(V_{i,\tilde{K_a}}) \qquad \text{(Modèle de prédiction)}$$

Cette attaque ne peut être menée que si les *offsets* et le *Shuffle* sont connus. L'attaque consiste d'abord en une première étape qui retrouve les *offsets* et contourne le *Shuffle*.

4.2 Se passer du *Shuffle*

L'opération des boîtes S sur 16 octets se décompose en 16 boîtes S traitées séquentiellement. Le *Shuffle* est une contre-mesure qui rend l'ordre de leur traitement aléatoire. C'est-à-dire que lorsqu'on veut obtenir la consommation à la sortie de la boîte S pour l'octet a, il faut regarder à l'instant correspondant au $Shuffle[a]^{ieme}$ calcul de boîte S. La valeur du *Shuffle* étant inconnue, il est difficile de connaître l'instant précis de traitement. En moyennant toutes les valeurs aux 16 instants de traitement, le bruit est trop important.

1. La première idée a été de retrouver le poids de Hamming de $Shuffle[a]$ avec une méthode SPA et de sommer les consommations aux b^{iemes} instants d'interêt où $HW(Shuffle[b]) = HW(Shuffle[a])$. Les résultats de cette méthode étaient diminués lorsque le poids de Hamming était égal à 2 (6 consommations à sommer).
2. Une seconde idée consiste à attaquer au niveau du Add Round Keys qui se fait dans l'ordre. Cependant, la CPA est moins efficace à cause de la linéarité entre la clé et la valeur intermédiaire.
3. La dernière idée a été de regarder si la sortie des boites S est réutilisée dans l'ordre à un autre moment. En fait, une fois les boites S calculées, lors des opérations suivantes (*Shift Rows, Mix columns*, compensation de masque), les octets sont de nouveau traités dans l'ordre. Les valeurs de sortie des boîtes S sont donc manipulées après l'application du shuffle dans le bon ordre. On peut donc, grâce à un apprentissage, connaître les instants où la consommation est corrélée à la valeur de sortie de la boîte S sans avoir à retrouver le *Shuffle*. Les instants trouvés correspondent au *Mix Columns*. Celui-ci utilise les valeurs de l'état dans un ordre fixe. La correlation est certes moins forte que lorsqu'on vise la sortie de la boîte S en connaissant le *Shuffle*, mais elle est suffisante pour l'attaque CPA.

4.3 Retrouver les *offsets*

Pour retrouver les *offsets*, on utilise une SPA. En d'autres termes, on essaye de trouver la valeur des *offsets* en déduisant de l'information des valeurs de consommation au sein de chaque trace. En début de chiffrement, les 16 *offsets* sont générés indépendamment. Ils sont ensuite incrémentés à chaque fois qu'une boite S est calculée. Pour chaque octet, **il y a 10 instants pour lesquels la consommation dépend de l'*offset***.

Attaque profilée

Une **attaque profilée** ou **attaque par template** est une attaque qui utilise un phase d'apprentissage sur un échantillon de courbes dont on connait les *offsets* et *Shuffle*.
Au départ, nous pensions répertorier la valeur moyenne de la consommation à l'instant de manipulation de l'*offset*. Ainsi, en lisant une consommation pour un *offset* inconnu, on associait la valeur répértoriée dont la moyenne associée s'approche le plus de la consommation.

Après une analyse plus précise de dépendances, l'*offset* dépend plus particulièrement non pas de la moyenne mais de la forme de l'onde de consommation au moment de sa manipulation. J'ai utilisé non plus les moyennes mais les spectres de Fourier moyens. On a alors un template d'apprentissage qui stocke des spectres de Fourier pour chaque tour et pour chaque octet. Cette méthode a permis de retrouver l'*offset* avec **99% de réussite**. L'inconvénient d'une attaque profilée est qu'en pratique, il est souvent impossible d'utiliser des courbes d'apprentissage où toutes les valeurs intermédiaires sont connues.

Attaque non profilée

Une attaque non profilée n'utilise pas d'apprentissage. Nous avons remarqué que la consommation augmentait avec le poids de Hamming de la valeur manipulée. Pour chaque octet, l'idée a été de stocker la suite des 10 consommations (après normalisation) pour chaque tour. Puis, en étudiant les valeurs, on peut retrouver

l'*offset*. Par exemple, pour l'*offset* de l'octet 0, on obtient

Tour	0	1	2	3	4	5	6	7	8	9	10
Consommation	200	190	254	**500**	**-5**	36	59	80	125	165	189

Ces valeurs sont reliées au poids de Hamming de l'*offset*. On repère ensuite le tour pour lequel l'écart est le plus fort entre deux valeurs consécutives. C'est sans doute le tour où l'*offset* passe de 15 (de poids de Hamming 4) à 0 (de poids de Hamming 0). Ici par exemple c'est entre les tours 3 et 4. L'*offset* deviné pour l'octet 0 est donc 16 - 4 = 12.
Avec cette méthode, on ne peut pas dépasser une réussite de 70%, en effet, dans 5/16 = 30% des cas, le passage de 15 à 0 ne s'effectue pas. En pratique, on déduit l'*offset* avec un **taux de réussite de 60%**.

4.4 Résultats

Etape de force brute

Une fois l'attaque effectuée, on obtient un classement des octets de clés du plus probable au moins probable. Pour améliorer l'attaque, on ajoute à la fin une étape de force brute. Il s'agit de tester la correspondance clair/chiffré avec la clé obtenue. Pour passer d'une hypothèse de clé à l'autre, on modifie les octets de clés du plus probable au moins probable.

Statistiques du DPA Contest

Pour pouvoir comparer les résultats avec ceux des autres attaques publiées sur le site, on utilise un *wrapper* fourni sur le site du DPA Contest qui permet d'évaluer les performances de l'attaque. Il est possible de tester l'attaque sur une base de données publique. Il est aussi possible de faire évaluer l'attaque par les organisateurs sur la base de données secrète en vue de publier les résultats sur la plateforme. Les résultats sont rassemblés dans la table 4.

TABLE 4 – Résultats des attaques du DPA Contest V4.2

Type	GSR>80%	Max PGE<10	GSR @1000%	Time/Trace (ms)
Profilée (sur mon PC)	28	43	1	341
Non Profilée (sur mon PC)	167	176	1	980
Non Profilée (testée par le DPA Contest)	**188**	**206**	1	**1000**
Non Profilée (Record du DPA Contest)	175	190	1	3500

Explication des statistiques :

— **GSR > 80%** : Nombre de traces nécessaires pour avoir un succès global au dessus de 80%
— **Max PGE < 10** : Nombre de traces nécessaires pour avoir une entropie (éloignement du bon octet de clé dans la liste des hypothèses) maximale au dessous de 10
— **GSR @1000** : Succès global après 1000 traces
— **Time/Trace** : Temps moyen par trace sur le PC de développement (Le mien est un intel i3-2120 3.30Ghz)

L'attaque non profilée utilise un nombre de traces légèrement au dessus de la meilleure attaque (en 175 traces) mais le temps moyen par trace est très inférieur à celui de la meilleure attaque (3,500 ms) [5]. Elle a été publiée sur le site du DPA Contest [6]. En ce qui concerne l'attaque profilée, d'autres attaques 100% *machine learning* ont des résultats très efficaces en une seule trace avec un temps de 100ms. L'inconvénient du *machine learning* est que les attaques ne seront pas forcément transférables sur d'autres cartes.

Conclusion

Les contre-mesures proposées par le DPA Contest V4.2 ont un fort potentiel mais ont certaines faiblesses. Il est possible de retrouver l'*offset* de manière SPA et de contourner le *Shuffle*. Ce dernier est une contre-mesure puissante mais trop localisée. L'*offset* est très souvent ré-utilisé et on peut combiner ces fuites en faisant une attaque horizontale pour récupérer sa valeur.

5 Proposition d'amélioration

Les failles exploitées dans l'AES de la partie précédente résident dans la possibilité de retrouver l'*offset* et de se passer du *shuffle*. Les brèves explications des attaques publiées sur la plateforme sous-entendent qu'elles aussi ont utilisé ces failles. Il est donc intéressant de chercher à contrer ces fuites en modifiant l'algorithme. La difficulté de cette partie a été de trouver un compromis entre l'efficacité et la sécurité de l'algorithme. J'étais souvent tentée d'améliorer fortement la sécurité sans penser que le stockage et le nombre de cycles exploseraient.

5.1 Forcer le *Shuffle* pour toutes les opérations de calcul

Dans l'implémentation, le *Shuffle* consiste à randomiser l'ordre des boites S. Mais avant et après leur application, les octets sont traités dans l'ordre. C'est grâce à cette faille que le système laisse fuir de l'information sur la sortie des boîtes S. Le *Mix Columns* suivant la première boîte S traite les octets de l'état dans un ordre fixé et fuit donc la valeur sensible.

Pour empêcher cette faille, l'idée initiale a été de garder le *Shuffle* comme ordre général de traitement des octets lorsque c'était possible. On modifie donc généralement l'ordre de calcul pour toutes les étapes de l'AES. Cette idée s'est avérée trop coûteuse en cycles. **Nous avons finalement choisi de ne protéger que les 3 permiers et les 3 derniers tours**. Le *Mix Columns*, qui était la partie qui fuyait le plus, a été modifié pour traiter les rangs de manière aléatoire. Cette méthode était en effet moins coûteuse que de randomiser l'ordre des octets.

5.2 Complexifier l'accès à l'*offset*

Nous sommes partis du principe que lorsque l'algorithme manipule un *offset*, le poids de Hamming de celui-ci est vulnérable. **Si l'on oblige la valeur manipulée à avoir un poids de Hamming constant, l'algorithme ne peut plus fuir d'information sur l'*offset***. Nous nous sommes donc inspirés de l'idée des *codes constant weight* pour l'appliquer à l'*offset*.

Définition 4. Code *Constant Weight : Un code de poids constant (constant weight) est un code dont tous les mots ont un poids de Hamming constant. On appelle (x,y)-code un code de poids constant égal à x sur y bits. Il contient* $\binom{x}{y}$ *éléments.*

L'idée de l'implémentation *constant weight* introduite en 2014 [15] est de remplacer l'ensemble les valeurs sensibles à manipuler par un (x,y)-code quitte à augmenter la taille de stockage. Cette idée est venue de

5. Après ma publication, une nouvelle attaque d'ordre supérieur a été publiée utilisant seulement 14 traces. Mon attaque arrive finalement en 3ème position

6. `http://www.dpacontest.org/v4/42_hall_of_fame.php`

Servant *et al.* qui ont proposé un calcul d'AES à poids constant. Dans cette implémentation, toutes les valeurs manipulées ont un poids constant. Le masquage en devient obsolète si le modèle de fuite est uniquement en poids de Hamming. La taille du code choisi pour remplaçer les 256 valeurs possibles a été le (5,11)-code de taille 462. Cette idée implique une forte augmentation du nombre de cycles à cause des opérations appliquées (boîtes S, ou xor par exemple) et de la taille de code pour le stockage des variables. Par contre, aucun aléa n'est généré. Les contraintes en stockage et en nombre de cycles n'ont pas permis d'appliquer cette contre-mesure dans notre contexte. Mais, **il a été très intéressant de s'en inspirer pour l'utiliser seulement pour des petites valeurs (les *offsets*). En effet, pour les valeurs de l'*offset* dont la seule opération appliquée est l'incrémentation, le poids constant n'est pas beaucoup plus coûteux et très efficace.**

On cherche donc un code de poids constant de taille minimale contenant au moins 16 valeurs. Il s'agit du (3,6)-code qui contient 20 élements. On associe un mot de code pour chaque indice comme dans la table 5. On définit alors un ordre arbitraire entre les mots de codes. Il est impératif de ne plus utiliser les indices mais seulement les mots de codes.

TABLE 5 – Mots de codes remplaçant les *offsets*

$0 \rightarrow 000111$	$2 \rightarrow 010011$	$9 \rightarrow 011010$	$11 \rightarrow 100110$
$15 \rightarrow 001011$	$14 \rightarrow 010101$	$5 \rightarrow 011100$	$13 \rightarrow 101001$
$6 \rightarrow 001101$	$12 \rightarrow 010110$	$7 \rightarrow 100011$	$4 \rightarrow 101010$
$10 \rightarrow 001110$	$3 \rightarrow 011001$	$1 \rightarrow 100101$	$8 \rightarrow 101100$

Le fait d'utiliser les mots de code dans l'ordre croissant est légèrement corrélé aux indices. C'est pourquoi les indices ont été associés aléatoirement. Les mots de codes remplacent définitivement les indices de 0 à 15. Pour faire fonctionner cette contre-mesure, il faut :

- créer une **fonction qui permet de passer d'un *offset* au suivant sans repasser par les indices**.
- modifier la définition du tableau des masques pour élargir les valeurs possibles des indices.
- créer une fonction qui génére les *offsets* sous forme de mots de codes. Cette fonction sera implémentée en C hors assembleur.

5.3 Mélanger les masques

Les masques définis par le DPA Contest V4.2 ont été étudiés pour contrer les attaques d'ordre 1 et 2. Cependant, le fait qu'ils soient rangés dans l'ordre croissant est légèrement lié à la valeur de l'*offset*. J'ai alors mélangé l'ordre de ces masques. Cela ne change pas l'efficacité de ce jeu de masques.

5.4 La contre-mesure proposée et son efficacité

La contrainte qui a conduit à ces modifications a été de prendre en compte le temps de calcul du schéma. Il n'était pas envisageable d'ajouter des bits d'aléa, ni d'augmenter significativement le nombre de cycles. En prenant en compte ces critères, le schéma obtenu a un temps de calcul similaire à celui du DPA Contest V4.2. **Le nombre de cycles a été augmenté de 6% et aucun autre aléa n'est tiré**. En intégrant ces idées, le pseudo-code modifié est placé en annexe C.

Conclusion

Le mélange des disciplines a été très fort dans cette partie (algorithmique, système d'exploitation, théorie des codes). **L'idée de l'application du constant weight pour l'*offset* paraît prometteuse pour contrer les fuites de l'AES RSM avec *Shuffle*.**

6 Evaluation du schéma proposé

La contre-mesure proposée n'a jamais été implémentée. Il n'y a donc pas de traces disponibles. Pour les obtenir et ainsi tester les attaques, j'ai utilisé un outil permettant de générer les traces de consommation sans avoir à faire des branchements électroniques : le ChipWhisperer (présenté dans l'article [12]). Cet outil a été crée par Colin O'Flynn et mis en vente il y a quelques années. Après un bon mois de tatonnements, j'ai réussi à le prendre en main et à faire des attaques sur le nouveau schéma. Les fuites m'ont parues comblées et l'attaque d'ordre 1 n'était plus envisageable même avec un très grand nombre de traces. Ce nouveau schéma a été proposé publiquement pour être soumis à d'autres attaquants. Cela m'a permis de remporter un prix à la conférence CHES 2016.

6.1 Traces de Consommation : données par le ChipWhisperer

Présentation du ChipWhisperer

La prise en main du ChipWhisperer a été difficile. Cet outil pédagogique a été conçu il y a plusieurs années et les outils software permettant de le manipuler sont fréquemment modifiés. La plupart de ces outils sont conçus pour Windows et pour des versions plus récentes du dispositif. Mon ordinateur de développement étant sous Debian, j'ai dû me plonger pendant plusieurs semaines dans l'architecture GNU/Linux pour résoudre les problèmes d'installations, de dépendances et de compatibilités.

Le Chipwhisperer est un outil permettant 3 actions :

1. La programmation d'algorithmes sur une puce, la *Victim board* (en bas à gauche sur l'image 9).
2. La mesure de traces de consommation de cette puce grâce au boitier *chipwhisperer capture* relié à un logiciel sur ordinateur (en bas à droite sur l'image 9).
3. L'attaque de ces traces enregistrées sur ordinateur via un logiciel reprogrammable : ChipWhisperer Analyser

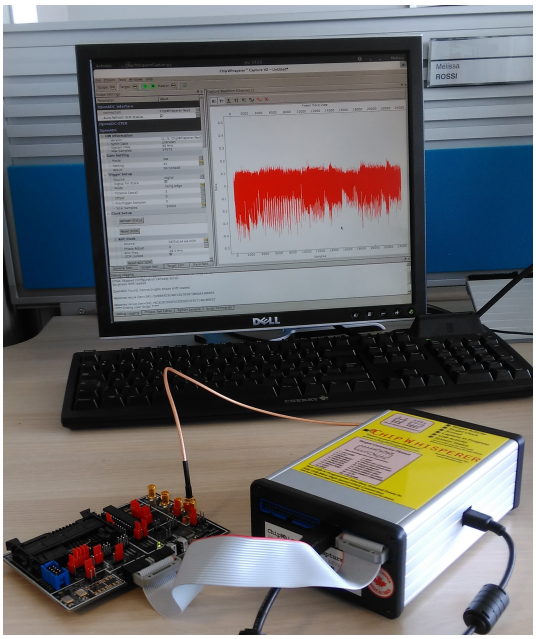

FIGURE 9 – Dispositif de mesure de traces

Une fois l'outil installé, il fallait programmer la puce *Victim Board* (une carte ATmega328P) pour qu'elle execute un AES avec des contre-mesures en assembleur. L'idée était de mettre le code du DPA Contest V4.2 et de le modifier petit à petit pour évaluer la sécurité des contre-mesures proposées. Mais, ce code n'était pas implémenté pour la même puce (une carte ATmega163), il a donc fallu modifier le code assembleur pour le rendre compatible. N'ayant pas suivi de cours d'électronique, j'ai eu des difficultés pour comprendre le fonctionnement du code assembleur et pourquoi il ne fonctionnait pas sur la victim board. J'ai été bloquée pendant plusieurs semaines avant de finalement comprendre qu'il fallait modifier toutes les adresses mémoires.

Comparaison du rapport signal à bruit

L'efficacité des attaques dépend fortement du dispositif utilisé. Il n'est pas possible de comparer directement les résultats obtenus sur ChipWhisperer avec ceux obtenus sur les traces du DPA Contest. Pour pouvoir comparer, il fallait appliquer les attaques des parties précédentes aux traces obtenues avec le ChipWhisperer. J'ai d'abord mesuré des traces de consommation pour l'**AES** du DPA contest V4.2. Les courbes mesurées m'ont paru plus bruitées que celles utilisées dans le DPA Contest. J'ai donc calculé le rapport signal sur bruit des courbes obtenues pour le comparer à celui du DPA Contest.

$$SNR = \frac{Var(P_{mesuree})}{Var(P_{bruits})} \qquad \text{(Rapport signal à bruit)}$$

Plus ce rapport est élevé, moins le signal est bruité. L'explication de l'obtention du SNR est en annexe D (expliqué aussi dans le livre [11]). J'ai obtenu qu'il y a environ 2 fois plus de bruit dans le ChipWhisperer que dans les courbes du DPA Contest. Par conséquent, il est normal que l'attaque sur Chipwhisperer nécessite au moins 2 fois plus de traces.
Finalement, **l'attaque du DPA Contest 4.2 sur ChipWhisperer a nécessité environ 3000 traces** au lieu des 1000 utilisées dans le chapitre 4 (sans force brute).

6.2 Attaque du schéma proposé

Une fois les contres-mesures développées en assembleur, j'ai tenté l'attaque du chapitre 4. On peut voir que les *offsets* sont complètement dissimulés. Les courbes en figures 10 et 11 montrent les coefficients de corrélation de l'*offset* et de la sortie de la boîte S en fonction du temps. Suite à cela, l'attaque présentée en chapitre 4, n'a pas fonctionné. Les résultats sont donnés dans la table 6. Même avec un nombre très important de traces, on ne retrouve aucun octet sur 16. La corrélation observée dans les courbes n'est pas totalement aléatoire et l'algorithme arrive à retrouver les instants d'interêts. Mais, le bruit est trop élevé pour arriver à déduire les bonnes hypothèses.

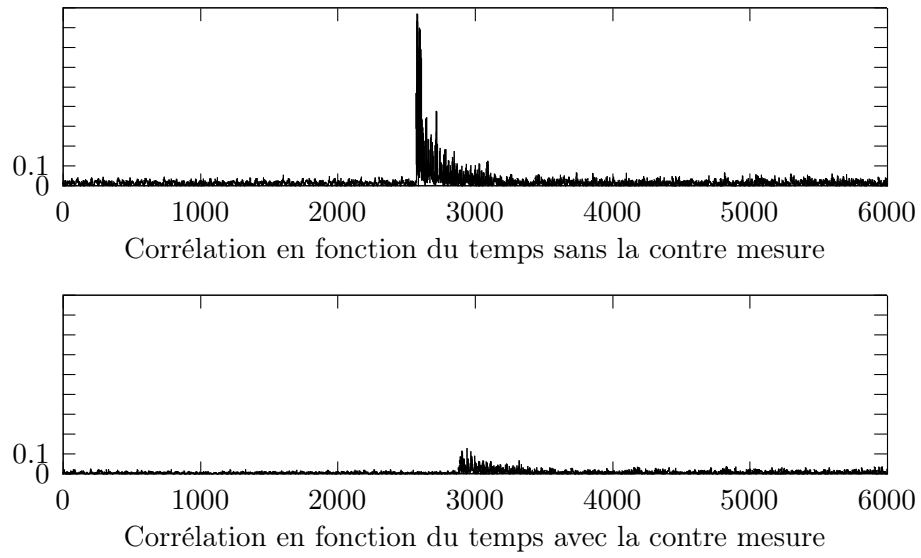

FIGURE 10 – Comparaison du coefficient de corrélation de l'*offset* avec et sans la contre-mesure

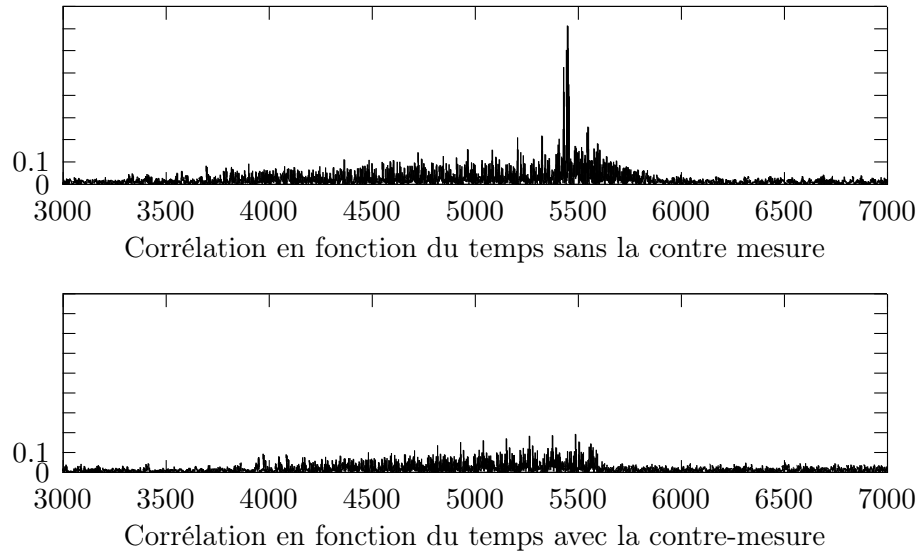

FIGURE 11 – Comparaison du coefficient de corrélation de la sortie de la boîte S avec et sans la contre-mesure

TABLE 6 – Résultats de l'attaque avec et sans contre-mesures

Version	Nombre de traces nécessaires pour retrouver 80% de la clé
Sans contre-mesures	3000
Avec contre-mesures	$>$ 15 000

6.3 Attaque sur des plateformes publiques

Le CHES Challenge est un concours international d'attaques lancé en juin 2016 où les participants peuvent eux-mêmes proposer des schémas d'AES. La phase de soumission s'est terminée le 31 juillet 2016. Les participants peuvent attaquer les schémas proposés et gagner des points. Ce concours s'achève fin août et les classements et les prix sont diffusés lors de la conférence CHES organisée à Santa Barbara. L'engouement pour ce concours a été très fort et de nombreux experts dans la recherche industrielle ou académique s'y sont lancés dans le but de décrocher le prix du meilleur attaquant. Publier sur cette plateforme est donc une idée efficace pour évaluer les performances du nouveau schéma.

Adapter les schémas aux exigences du challenge a été une tâche ardue puisqu'il fallait encore modifier le code pour qu'il s'adapte à une nouvelle carte (Xmega128D3). J'ai finalement réussi à publier la version du DPA Contest V4.2 avec et sans les contre-mesures sous le pseudonyme "Melissa (France)" [7]. Il n'y avait aucun doute sur le fait que mes schémas allaient être attaqués rapidement. En effet, la "durée de vie" d'un challenge sur cette plateforme est en moyenne de 24h. De plus, il n'y a pas de contrainte d'efficacité donc les challenges habituellement postés sont plus difficiles à résoudre. Dès le soir de ma publication, j'ai reçu plusieurs messages de chercheurs me disant qu'ils étaient en train d'attaquer mes schémas. Une fois les attaques menées, j'ai pu discuter avec eux et connaître les failles qu'ils ont exploitées. Celles-ci sont référencées dans la table 7.

7. `https://ctf.newae.com/flags/`

L'attaque menée sur la version avec la contre-mesure a utilisé **une faille qui n'est pas exploitable sur la version 4.2 du DPA Contest**. En effet, l'attaque consiste à créer des templates en fonction de la valeur de la clé au moment de son chargement. Pour cela, il faut des traces correspondant à plusieurs clés différentes. Celles-ci ne sont pas disponibles sur la plateforme du DPA Contest, il n'y a que les traces avec une clé fixée et plusieurs messages.

TABLE 7 – Résultats des attaques sur le CHES Challenge

Version	Temps avant d'être attaqué	Nombre d'attaquants	Failles exploitées
Sans amélioration	38h	4	*Offset* et *Shuffle* retrouvés par template et attaque CPA d'ordre 1
Avec amélioration	49h	3	Attaque par template au chargement de la clé **avant le chiffrement**

Finalement, les résultats du CHES Challenge sont plutôt encourageants quant à la robustesse de ces contre-mesures. Suite à ces résultats, j'ai été encouragée à participer à d'autres attaques. J'ai finalement réussi à attaquer 10 autres schémas publiés en utilisant des attaques CPA à différents instants, des resynchronisations et des apprentissages. Grâce à ces attaques et aux soumissions réalisées, j'ai pu remporter le prix *Best Student CHES Challenge 2016* qui a été décerné lors de la conférence annuelle CHES 2016. Ce prix m'a permis de recevoir un lot composé d'un ChipWhisperer Lite et un Arty FPGA Board.

Le schéma amélioré étant une suite directe du DPA contest V4.2, les membres du DPA Contest sont interessés par ce nouveau code. Les différences entre le processeur utilisé pour le DPA Contest et par le ChipWhisperer me demandent de réaliser quelques ajustements que j'ai déjà opérés dans le sens inverse. Après quoi, je soumettrai ma nouvelle version à l'équipe de Télécom en espérant qu'elle pourra être testée par la communauté.

7 Etude d'un schéma très sécurisé

Dans les parties précédentes, l'accent a été mis sur le nombre de cycles. Le travail de mon dernier mois de stage sera d'étudier un autre type de contre-mesure pour l'AES en se restreignant moins sur l'efficacité. Ce sénario correspond à des applications software où le nombre de cycles est moins important. Il s'agira de choisir une contre-mesure parmi celles proposées cette année à CHES dans [7] ou [1]. Ensuite, l'idée sera d'étudier sa sécurité grâce au ChipWhisperer.

Conclusion

En conclusion, j'espère que ce rapport aura donné un aperçu des contre-mesures par masquage et randomisation des attaques par consommation de puissance. La recherche est très dynamique sur le sujet et de nombreuses pistes restent à être étudiées, optimisées et combinées. Les idées à la base des contre-mesures abordées (masquage, randomisation, codes) ont un caractère général et peuvent être appliquées dans d'autres systèmes de chiffrement symétriques ou asymétriques. J'ai répertorié toutes les contre-mesures abordées dans un aide mémoire en annexe E.

Bibliographie

[1] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.

[2] Sonia Belaïd. *Security of Cryptosystems Against Power-Analysis Attacks.* Theses, ENS, October 2015.

[3] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the DPA contest v4 implementation. In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*, volume 8804 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2014.

[4] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany.

[5] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[6] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany.

[7] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software ? *IACR Cryptology ePrint Archive*, 2016 :264, 2016.

[8] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[9] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[10] Kerstin Lemke, Kai Schramm, and Christof Paar. DPA on n-bit sized Boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 205–219, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer, Heidelberg, Germany.

[11] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks : Revealing the Secrets of Smart Cards (Advances in Information Security).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[12] Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer : An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure*

Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers, volume 8622 of *Lecture Notes in Computer Science*, pages 243–260. Springer, 2014.

[13] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, California, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

[14] Wolfgang Rosenstiel and Lothar Thiele, editors. *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*. IEEE, 2012.

[15] Victor Servant, Nicolas Debande, Houssem Maghrebi, and Julien Bringer. Study of a novel software constant weight implementation. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2014.

Annexes

A Pseudo-code de l'attaque CPA

Algorithme 1 : Attaque au dernier tour du DPA Contest V2
Entrées : Traces du DPA Contest V2, a octet de clé ciblé
Sorties : K_a

```
/* Définition de l'intervalle d'étude des instants                    */
si Attaque aveugle alors
    E = [2350, 2450]
sinon
    E = 2370
/* Boucle sur les instants à étudier                                  */
pour t ∈ E faire
    /* Boucle sur les octets possibles de clé                         */
    pour 0 ≤ k ≤ 255 faire
        /* Boucle sur les traces                                      */
        pour 0 ≤ i ≤ 20000 faire
            Calculer H_{i,a,k} = HW(S^{-1}(C_{i,a} ⊕ k) ⊕ C_{i,SR^{-1}(a)})
            Lire W_{i,t} = W(i,t)
        Calculer ρ_{a,t,k} = Pearson((H_{i,a,k})_{0≤i≤20000}, (W_{i,t})_{0≤i≤20000})
    OctetCle_a(t) = argmax_k(ρ_{a,t,k})
    Compteur(OctetCle_a(t)) + +;
K_a = argmax_l(Comteur(l))
```

B Algorithme AES-RSM amélioré du DPA Contest V4.2

Algorithme 2 : AES-RSM amélioré du DPA Contest v4.2

Entrées : Clair sur 16 Octets X_0,..X_{15}

11 sous-clés $K_r = K_{r,0}...K_{r,10}$ de 16 octets obtenues avec l'algorithme de Key Schedule

Sorties : Chiffré sur 16 Octets X_0,..X_{15}

`/* Définition des masques */`

Masques = 0x03, 0x0c, 0x35, 0x3a, 0x50, 0x5f, 0x66, 0x69, 0x96, 0x99, 0xa0, 0xaf, 0xc5, 0xca, 0xf3, 0xfc

`/* Génération de 16 offsets aléatoires sur 4 bits */`

$offset[0] \leftarrow \$$

...

$offset[15] \leftarrow \$$

`/* Génération aléatoire de 2 permutations shuffle0 et shuffle10 */`

$Shuffle0[0...15] \leftarrow \$$

$Shuffle10[0...15] \leftarrow \$$

$K_0 \leftarrow K_0 + Masques[offset[]]$

pour $r \in [0, 9]$ **faire**
- $X \leftarrow X \oplus K_r$
- **si** r=*0* **alors**
  - **pour** $i \in [0, 15]$ **faire**
    - $X_{shuffle0[i]} \leftarrow BoiteSMasquee_{offset[shuffle0[i]]}(X_{shuffle0[i]})$
- **sinon**
  - **pour** $i \in [0, 15]$ **faire**
    - $X_i \leftarrow BoiteSMasquee_{offset[i]+r}(X_i)$
- $X \leftarrow ShiftRows[X]$
- $Y \leftarrow MixColumns[X]$
- **pour** $i \in [0, 15]$ **faire**
  - $CompensationMasque[i] =$
    $ShiftRows(MixColumns(Masque[Offset[i] + r + 1])) \oplus Masque(Offset[i] + r + 1)$
- $X \leftarrow X \oplus CompensationMasque[]$

`/* Dernier tour */`

$X \leftarrow X \oplus K_{10}$ **pour** $i \in [0, 15]$ **faire**
- $X_{shuffle10[i]} \leftarrow BoiteSMasquee_{offset[shuffle10[i]+10]}(X_{shuffle10[i]})$

$X \leftarrow ShiftRows[X]$ $X \leftarrow X \oplus K_{10}$

pour $i \in [0, 15]$ **faire**
- $CompensationMasqueDernierTour[i] =$
  $ShiftRows(Masque[Offset[i] + 10]) \oplus Masque(Offset[i] + 10)$

$X \leftarrow X \oplus CompensationMasqueDernierTour[]$

C AES-RSM amélioré du DPA Contest V4.2 avec contre-mesures

Voici les parties de code à ajouter :

Algorithme 3 : Modifications à ajouter pour compléxifier l'accès à l'offset

`/* Définition du tableau des masques */`

Script *GénérerTableauMasques* **:**

Stockage des Masques mélangés dans des adresses de poids constant. On choisit une adresse de la carte qui est libre : par exemple adr=0x0600.

(adr + 000111) ←0xaf	(adr + 010011) ← 0xa0	(adr + 011010) ← 0xf3	(adr + 100110) ← 0x69
(adr + 001011) ← 0x66	(adr + 010101)← 0x35	(adr + 011100) ← 0x99	(adr + 101001) ← 0xfc
(adr + 001101) ← 0xca	(adr + 010110) ← 0x0c	(adr + 100011) ← 0x5f	(adr + 101010) ← 0x03
(adr + 001110) ← 0xc5	(adr + 011001) ← 0x3a	(adr + 100101) ← 0x50	(adr + 101100) ← 0x96

Stockage d'un tableau "suivant" indiquant les adresses des offsets suivants. On choisit une autre adresse libre : par exemple adr2 = 0x500

(adr2 + 000111) ← 0x25	(adr2 + 010011) ← 0x19	(adr2 + 011010) ← 0x0e	(adr2 + 100110) ← 0x16
(adr2 + 001011) ← 0x07	(adr2 + 010101)← 0x0b	(adr2 + 011100) ← 0x0d	(adr2 + 101001) ←0x15
(adr2 + 001101) ← 0x23	(adr2 + 010110) ← 0x29	(adr2 + 100011) ← 0x2c	(adr2 + 101010) ← 0x1c
(adr2 + 001110) ← 0x26	(adr2 + 011001) ← 0x2a	(adr2 + 100101) ← 0x13	(adr2 + 101100) ← 0x1a

`/* Fonction de génération des offsets codée hors carte */`

Fonction *GenererOffset()* **:**

Sorties : 16 mots de code correspondant aux offsets : offset[0]...offset[15]
correspondance[0..15] = [7,37,19,25,42,28,13,35,44,26,14,38,22,41,21,11]
pour $0 \leq a \leq 15$ **faire**
  offset[a] ← Correspondance[$]
renvoyer offset

Fonction *Suivant(int offset,int r)* **:**

Entrées : Un offset
Sorties : Le mot de code correspondant à l'offset "incrémenté" r fois
pour $0 \leq i \leq r$ **faire**
  offset ← suivant[offset]
renvoyer offset

Et voici en page suivante le pseudo-code de l'algorithme AES-RSM modifié. Les modifications concernant l'offset sont en bleu. Les modifications concernant le shuffle sont en rouge. Lorsque les fonctions Add Round Keys et ShiftRows sont en rouge, l'ordre de traitement des octets est randomisé grâce au Shuffle. Lorsque la fonctions Mix Columns est en rouge, l'ordre de traitement des colonnes est randomisé grâce au Shuffle.

Algorithme 4 : AES-RSM du DPA Contest v4.2 modifié

Entrées : Clair sur 16 Octets X_0,..X_{15}
11 sous-clés $K_r = K_{r,0}...K_{r,10}$ de 16 octets obtenues avec l'algorithme de Key Schedule
Sorties : Chiffré sur 16 Octets X_0,..X_{15}

GénérerTableauMasques() ;
offset[]=GénérerOffsets ;

`/* Génération aléatoire de 2 permutations shuffle0 et shuffle10 */`
$Shuffle0[0...15] \leftarrow \$$
$Shuffle10[0...15] \leftarrow \$$

$K_0 \leftarrow K_0 + Masques[offset[]]$

pour $r \in [0, 9]$ **faire**
  $X \leftarrow X \oplus K_r$
  si $r \in [0, 1, 2, 8, 9]$ **alors**
    shuffle = shuffle0 pour les tours 0, 1 et 2
    shuffle = shuffle10 pour les tours 8 et 9
    pour $i \in [0, 15]$ **faire**
      $X_{shuffle[i]} \leftarrow BoiteSMasquee_{Suivant(offset[shuffle[i]],r)}(X_{shuffle0[i]})$
  sinon
    pour $i \in [0, 15]$ **faire**
      $X_i \leftarrow BoiteSMasquee_{Suivant(offset[i],r)}(X_i)$
  $X \leftarrow ShiftRows[X]$
  $Y \leftarrow MixColumns[X]$
  pour $i \in [0, 15]$ **faire**
    $CompensationMasque[i] = ShiftRows(MixColumns(Masque[Suivant(offset[i], r + 1)])) \oplus Masque(Suivant(offset[i], r + 1))$
  $X \leftarrow X \oplus CompensationMasque[]$
`/* Dernier tour */`
$X \leftarrow X \oplus K_{10}$ **pour** $i \in [0, 15]$ **faire**
  $X_{shuffle10[i]} \leftarrow BoiteSMasquee_{offset[shuffle10[i]+10]}(X_{shuffle10[i]})$
$X \leftarrow ShiftRows[X]$
$X \leftarrow X \oplus K_{10}$
pour $i \in [0, 15]$ **faire**
  $CompensationMasqueDernierTour[i] = ShiftRows(Masque[Suivant(offset[i], 10)]) \oplus Masque(Suivant(offset[i], 10))$
$X \leftarrow X \oplus CompensationMasqueDernierTour[]$

D Calcul du rapport signal à bruit

Voici la formule du rapport signal à bruit en fonction de la puissance mesurée :

$$SNR = \frac{Var(P_{mesuree})}{Var(P_{bruits})} \qquad \text{(Rapport signal à bruit)}$$

La variance de la puissance correspondant aux bruits est obtenue en calculant la variance pour une certaine selection de courbes. On ne choisit que les courbes ayant la même valeur de fuite. Par exemple, on calcule la valeur théorique pour en sortie de la boîte S. Si le poids de Hamming de cette valeur est égal à 1, on selectionne la courbe. Les SNR obtenus sont rassemblés dans la table 8.

HW	SNR CW	SNR DPA Contest	rapport DPAContest/CW
0	616	500000	326
1	54	91	1.70
2	15	18	1.2
3	8	9	1.1
4	6	7	1.16
5	8	9	1.1
6	16	20	1.25
7	72	87	1.20
8	981	4040	4.11

TABLE 8 – Comparaison des rapports signal à bruit

Pour des poids de Hamming égaux à 0 ou 8, on voit que le bruit est très faible. En effet, les valeurs des 8 bits sont fixées soit à 0 soit à 1. La moyenne pondérée des rapports entre les SNR du DPA Contest et du CW est de 2. Il y a donc en moyenne 2 fois plus de bruit dans le Chipwhisperer que dans les courbes du DPA Contest.

E Récapitulatif des contre-mesures abordées

Le tableau suivant présente une idée générale des contre-mesures rencontrées dans ce stage. Celles-ci peuvent être combinées, optimisées, simplifiées en fonction des besoin en sécurité/efficacité.

Contre-mesure	Description	Sécurité	Efficacité
Jitter	Contre-mesure **localisée** qui fait attendre un temps aléatoire avant une étape sensible choisie	Facile à contourner avec une resynchronisation	Coûteuse car cela demande la génération de grands entiers aléatoires
Shuffle	Contre-mesure **localisée** qui randomise l'ordre de traitement de l'état au moment d'une étape sensible choisie	Réduit efficacement les possibilités d'attaques CPA de tout ordre au niveau de l'étape sensible. Attention, il faut éviter les fuites lors de la génération d'aléa	Assez coûteuse car cela demande la génération d'une permutation aléatoire en précalcul et cela rallonge le code. Cette contre-mesure n'est envisageable que pour un petit nombre d'étapes.
Masquage booléen d'ordre d	Chaque donnée est démultipliée en $d+1$ masques. L'opération boîtes S est remplacée par une opération prenant en compte les masques (avec des tables pré-calculées, technique bitslice, ou décomposition en multiplications). Les nouveaux masques nécessitent une génération d'aléa	Très efficace, ce masquage permet d'éviter toutes les attaques verticales d'ordre d.	Très forte perte de vitesse.
Implémentation *constant weight*	Toutes les valeurs manipulées sont changées en valeurs de poids de Hamming constant	Empêche toutes les attaques CPA à tout ordre dans le modèle de fuites en poids de Hamming.	Entraine une forte perte de vitesse mais aucun aléa n'est généré
Masquage RSM	Masquage d'ordre 1 avec un ensemble fixe de masques tournants	Empêche les attaques d'ordre 1 mais la manipulation de l'indice du masque tournant peut entraîner des attaques horizontales	Peu coûteux